TUGAS AKHIR – KI141502

IMPLEMENTASI VIRTUAL DATA CENTER MENGGUNAKAN LINUX CONTAINER BERBASIS DOCKER DAN SDN

MUHAMMAD FIKRI ALAUDDIN
NRP 5113100068

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI VIRTUAL DATA CENTER MENGGUNAKAN LINUX CONTAINER BERBASIS DOCKER DAN SDN

MUHAMMAD FIKRI ALAUDDIN
NRP 5113100068

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

VIRTUAL DATA CENTER IMPLEMENTATION USING DOCKER BASED LINUX CONTAINER AND SDN

MUHAMMAD FIKRI ALAUDDIN
NRP 5113100068

First Advisor
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Second Advisor
Ir. Muchammad Husni, M.Kom.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI VIRTUAL DATA CENTER MENGGUNAKAN LINUX CONTAINER BERBASIS DOCKER DAN SDN

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
MUHAMMAD FIKRI ALAUDDIN
NRP : 5113100068

Disetujui oleh Pembimbing Tugas Akhir:

1. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
   NIP: 19770824 200604 1 001 ..... (Pembimbing 1)
2. Ir. Muchammad Husni, M.Kom.
   NIP: 19600221 198403 1 001 ..... (Pembimbing 2)

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI INFORMASI
DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA
JUNI, 2017

(Halaman ini sengaja dikosongkan)

`

IMPLEMENTASI VIRTUAL DATA CENTER MENGGUNAKAN LINUX CONTAINER BERBASIS DOCKER DAN SDN

Nama Mahasiswa : MUHAMMAD FIKRI ALAUDDIN

NRP : 5113100068

Jurusan : Teknik Informatika FTIF-ITS

Dosen Pembimbing 1 : Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing 2 : Ir. Muchammad Husni, M.Kom.

ABSTRAK

Software Defined Networking (SDN) adalah teknik jaringan yang berbasis aplikasi sehingga pengguna hanya tinggal menjalankan saja dan software akan menghandle konfigurasi yang diperlukan, tidak seperti teknik jaringan konvensional yang memerlukan konfigurasi dari admin untuk masalah seperti routing. Oleh karena itu sangat cocok untuk diterapkan di Data Center berbasis virtual yang membutuhkan trafik yang sangat fleksible dan akses jaringan yang on-demand.

Di era modern ini, teknologi jaringan dituntut untuk akses yang semakin cepat dan semakin dinamis untuk memenuhi kebutuhan dari pengguna yang semakin banyak. Sama halnya dengan Data Center. Oleh karena itu Tugas Akhir ini mengimplementasi kan Data Center berbasis virtual menggunakan Linux Container bebasis Docker dan SDN untuk menjawab akan permintaan akses yang dinamis dan on-demand.

Pada tugas akhir ini penggunaan Docker sebagai host dari Data Center berbasis virtual dinilai berhasil dan memang cocok untuk diterapkan. Dengan hasil performa yang dibilang berhasil dan dapat digunakan dan berjalan dengan baik. Docker terintegrasi dengan baik kedalam switch jaringan Data Center berbasis virtual yang dikontrol dengan controller yang berbasis dan menggunakan teknologi SDN.

Kata kunci: Software Defined Networking, Data Center berbasis Virtual, Software Container, Docker.

`

VIRTUAL DATA CENTER IMPLEMENTATION USING DOCKER BASED LINUX CONTAINER AND SDN

Student's Name	**: MUHAMMAD FIKRI ALAUDDIN**
Student's ID	**: 5113100068**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**
Second Advisor	**: Ir. Muchammad Husni, M.Kom.**

ABSTRACT

Software Defined Networking or SDN is networking technique that is application based so that what user need to do is just running it and the software will handling configuration that is needed, unlike conventional networking technique that need manual configuration from admin for problem like routing. So SDN is very fitting to be used in Virtual Data Center that need highly flexible traffic and on-demand access to the network.

In this modern era, networking technology is needed for better and faster access and needed to be more dynamic for fulfill the demand from the fastly increasing number of user. Same with Data Center. So this final project is implementing Virtual Data Center using Docker based Linux Container and SDN to answer the need for dynamic and on-demand access.

In this undergraduate thesis, the usage of Docker as the host of the Virtual Data Center considered working and fit to use at the Virtual Data Center system itself, with satisfying bechmarking

result and can run and work properly. Docker is integrated well with the switch in the networking system which is controlled by controller that is based and use SDN technology

Keywords : Software Defined Networking, Virtual Data Center, Software Container, Docker.

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya dan junjungan Nabi Muhammad SAW sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**IMPLEMENTASI VIRTUAL DATA CENTER MENGGUNAKAN LINUX CONTAINER BERBASIS DOCKER DAN SDN**"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak Budi Hariono dan Ibu Imas Parnika Winata selaku orang tua penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D selaku pembimbing I sekaligus dosen wali yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
3. Ir. Muchammad Husni, M.Kom. selaku II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS.
5. Bapak Dr. Radityo Anggoro, S.Kom.,M.Sc. selaku koordinator S1 di Jurusan Teknik Informatika ITS.

6. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
7. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
8. Jasmine Amelia Ulfah selaku pemberi motivasi dan orang yang selalu ada untuk penulis.
9. Muhammad Luthfi Fakruddin dan Syaidah Azzura Ramadhani selaku saudara yang selalu mendukung dalam pengerjaan Tugas Akhir ini.
10. Dhanar Prayoga selaku teman seperjuangan, kamerad, dalam pengerjaan Tugas Akhir ini.
11. Teman-teman seperjuangan RMK NCC/KBJ dan KCV, yang telah menemani dan menyemangati penulis.
12. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
13. Teman-teman angkatan 2013, yang sudah mendukung saya selama perkuliahan.
14. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Mei 2017

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR PSEUDOCODE

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Adanya ledakan jumlah dari perangkat mobile dan kontennya, virtualisasi server, dan digembar gemborkannya teknologi *cloud computing* adalah beberapa alasan kenapa sangat dibutuhkannya arsitektur atau teknik networking baru yang dimana bisa mengatasi keinginan dari teknologi baru yang disebutkan diatas. Struktur network statis yang lawas mulai tidak dapat mengatasi permintaan teknologi terbaru yang sudah memakai komputasi dinamis yang membutuhkan *data storage* yang besar.

Disini penulis mencoba untuk mengaplikasikan salah satu teknik arsitektur terbaru yaitu SDN (*Software-defined Networking).* SDN adalah teknik jaringan yang berbasis aplikasi sehingga pengguna hanya tinggal menjalankan saja dan software akan menghandle konfigurasi yang diperlukan, tidak seperti teknik jaringan konvensional yang memerlukan konfigurasi dari admin untuk masalah seperti routing.

Untuk mempermudah vitualisasi jaringan penulis menggunakan Docker sebagai *host* dari mininet yang berperan sebagai *container* untuk mempermudah mengelola dan mempermudah konfigurasi dari jaringan sendiri.

Kenapa penulis memlih menggunakan SDN? Karena SDN menjawab permasalahan bahwa arsitektur statis dari jaringan konvensional sangat tidak cocok untuk diterapkan di dalam komputasi dan penyimpanan dinamis yang dimana dibutuhkan di *data center,* kampus, dan lingkungan kerja dewasa ini. Kunci masalah dari dibutuhkannya arsitektur jaringan baru yaitu :

- Perubahan pola trafik: Banyak aplikasi sekarang harus mengakses database dan server yang berada di tempat berbeda lewat *cloud.* Hal ini menyebabkan diharuskannya managemen trafik yang sangat fleksibel dan akses jaringan *on-demand.*

- Munculnya layanan *cloud*: User menginginkan akses kepada aplikasi, infrastruktur dan sumber daya IT yang *on-demand.*

Keuntungan yang didapat dari SDN yaitu dapat diprogram secara langsung, mudah beradaptasi, dapat dikelola secara terpusat, dapat dikonfigurasi langsung lewat program, dan settingan seragam antar jaringan sehingga cocok untuk menjawab permasalah diatas. SDN juga sangat menghemat waktu, menyerdehanakan konfigurasi yang ada, dan mengurangi kemungkinan untuk mesin melakukan error. Di dunia yang serba cepat mudah dan praktis tentunya teknik ini sangat cocok diimplementasikan dibandingkan dengan teknik teknik konvesional biasa.

Penulis memilih untuk menggunakan *software* virtualisasi jaringan berupa mininet untuk pengaplikasian *Virtual Data Center* berbasis SDN ini dikarenakan pertama, murah karena tidak diperlukannya alat tambahan, dan juga dapat merefleksikan jaringan fisik dengan sama persis karena konfigurasi dan kodingan didalam Mininet dibuat semirip mungkin dengan konfigurasi jaringan di dunia nyata.

Penulis juga memilih menggunakan *container* sebagai *host* untuk lebih mempresentasikan permasalahan permasalahan arsitektur jaringan yang telah dijabarkan sebelumnya. Dipilihnya Docker sebagai *container* karena popular dengan dukungan yang besar.

1.2 Rumusan Masalah

Rumusan masalah yang berusaha diselesaikan dalam tugas akhir ini adalah :

1. Bagaimana mengimplementasikan dan mengelola Docker sebagai infrastruktur *Virtual Data Center* ?
2. Bagaimana mengimplementasikan SDN untuk pengelolaan *Virtual Data Center* berbasis Docker ?

3. Bagaimana mengevaluasi kapasitas *Virtual Data Center*?
4. Bagaimana *provisioning* server berbasis aplikasi dalam *Virtual Data Center* berbasis Docker dan SDN ?

1.3 Batasan Permasalahan

Pada tugas akhir ini ada beberapa batasan yang menjadi pertimbangan, berikut ini batasan-batasan yang menjadi pertimbangan :

1. Dibangun diatas platform virtualisasi Mininet .
2. Menggunakan bahasa pemrograman Python.
3. Mininet *host* menggunakan *container* Docker.

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah :

1. Membuat infrastruktur *virtual data center* berbasis Docker dengan teknik jaringan SDN.
2. Mengetahui kinerja dari *virtual data center* berbasis Docker dengan teknik jaringan SDN.

1.5 Manfaat

Pengerjaan tugas akhir ini memiliki manfaat untuk mempermudah network admininstrator mengkonfigurasi jaringan di lingkungan yang fluktuatif.

1.6 Metodologi

Berikut ini metodologi yang diterapkan dalam pengerjaan tugas akhir :

1.6.1 Penyusunan Proposal

Tahap awal pengerjaan tugas akhir ini dimulai dengan penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir ini berisi tentang perencanaan sistem *virtual data center,* seperti platform

apa yang digunakan, program progam apa saja yang dirasa cocok oleh penulis sebagai dasar untuk membangun *virtual data center* berbasis SDN.

Proposal Tugas Akhir ini terdiri dari deskripsi pendahuluan yang berisi penjabaran latar belakang dan rumusan masalah yang menjadi dasar pengerjaan tugas akhir, batasan masalah dalam perancangan jaringan, serta tujuan dan manfaat yang diharapkan dapat tercapai dengan perancangan sistem *virtual data center* ini. Pada proposal Tugas Akhir ini juga menyertakan tinjauan pustaka yang menjelaskan berbagai teori yang menjadi dasar pembuatan tugas akhir ini, yaitu sistem *virtual data center* menggunakan kontainer linux berbasis docker dengan SDN *controller.*

1.6.2 Studi Literatur

Studi literatur yang dilakukan dalam pengerjaan Tugas Akhir ini dilakukan dengan mencari informasi dari sumber-sumber terkait yang membahas mengenai perancangan sistem *virtual data center* menggunakan kontainer linux berbasis docker dengan SDN *controller*. Selain itu, juga dilakukan studi literatur mengenai penerapan sistem *vitual data center* kepada GUI berbasis web menggunakan HTML dan Flask.

Hal-hal lain yang dipelajari dalam studi literatur meliputi Mininet *virtual network*, *controller* berbasis SDN, pemrograman Python, sistem Docker, pemrograman web dan lain sebagainya.

1.6.3 Analisis dan Desain Perangkat Lunak

Tahap ini meliputi perancangan dan analisis sistem berdasarkan studi literatur. Berdasarkan konsep teknologi dari perangkat lunak yang ada saat ini, dilakukan perancangan sistem yang akan dibangun. Langkah-langkah pengerjaan juga didefinisikan pada tahap ini.

Pada tahapan ini dibuat prototipe dari sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Selain itu, dalam tahap ini juga dilakukan desain sistem dan proses-proses yang ada.

1.6.4 Implementasi Perangkat Lunak

Implementasi Perangkat Lunak merupakan tahapan untuk membangun rancangan program yang telah dibuat. Pada tahapan ini merealisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

1.6.5 Pengujian dan Evaluasi

Pada tahapan ini dilakukan uji coba pada alat yang telah dibuat. Tahapan ini dimaksudkan untuk mengevaluasi tingkat akurasi dan performa dari alat tersebut serta mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

1.6.6 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, dasar teori, implementasi, serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan

Penulisam buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir scara menyeluruh. Selain itu, diharapkan dapat bermanfaat bagi pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.

2. Bab II. Tinjauan Pustaka
   Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi implementasi dari perancangan perangkat lunak yang telah dibuat pada bab sebelumnya.

4. Bab IV. Implementasi
   Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *pseudocode* yang digunakan untuk proses implementasi.

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.

8. Lampiran
   Dalam lampiran terdapat kode sumber dari program secara keseluruhan.

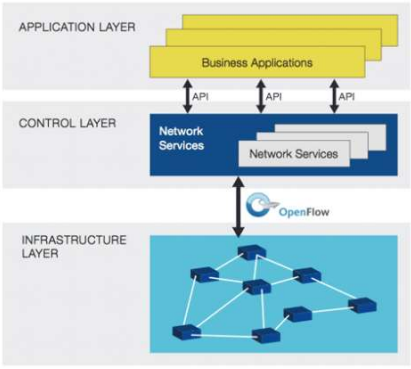
(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

Pada bab ini, dijabarkan tentang penjelasan teori-teori yang berkaitan dengan pokok bahasan tugas akhir. Bab ini juga menjelaskan modul dan alat yang nantinya akan digunakan pada tahap implementasi program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap alat yang digunakan dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 *Software-Defined Networking (SDN)*

Software-Defined Networking (SDN) atau Jaringan Berbasis Aplikasi adalah pemisah fisik dari *network control plane* terhadap *forwarding plane* dimana *control plane* menaungi atau mengontrol beberapa alat[1]. SDN memungkinkan pemisahan antara *control plane* dan *data plane* pada sebuah jaringan, sehingga admin network dapat melakukan konfigurasi pada sebuah terminal tanpa perlu menyentuh masing masing *switch.* SDN dibangun berdasarkan protokol OpenFlow. Gambar 2.1 merupakan ilustrasi atau gambaran mengenai penerapan SDN.



Gambar 2.1 Contoh Ilustrasi Penerapan SDN[1]

Keuntungan yang didapat dari SDN yaitu dapat diprogram secara langsung, mudah beradaptasi, dapat dikelola secara terpusat, dapat dikonfigurasi langsung lewat program, dan settingan seragam antar jaringan.

2.2 OpenFlow

OpenFlow adalah standart terbuka yang memungkinkan para peneliti untuk menjalankan eksperimen protokol di jaringan kampus yang kita gunakan setiap hari[2].

Router atau switch konvensional *data path* dan *control path* dilakukan didalam alat yang sama. OpenFlow switch memisahkan kedua fungsi tersebut sehingga *data path* tetap berada di switch, sedangkan *high-level routing decisions* dipindah ke *controller* terpisah yang biasanya berupa server. *Switch* dan *controller* berkomunikasi melalui protokol OpenFlow yang dimana mendefinisikan pesan seperti *packet-received, send-packet-out, modify-forwarding-table,* dan *get-stats.*

OpenFlow digunakan di aplikasi seperti *virtual machine*, jaringan dengan sekuritas tinggi, dan jaringan *mobile* berbasis ip generasi baru.

2.3 Virtualisasi Jaringan

Virtualisasi adalah kemampuan untuk mensimulasikan platform hardware, seperti server, media penyimpanan maupun perangkat jaringan didalam perangkat lunak[3]. Semua fungsi tersebut dipisahkan dari perangkat keras dan disimulasikan sebagai sebuah "*virtual instance"*, dengan kemampuan beroperasi layaknya perangkat keras tradisional.

Virtualisasi Jaringan didefiniskan sebagau kemampuan untuk menciptakan network virual logis yang dipisahkan dari jaringan dasar perangkat keras untuk menjamin jaringan dapat diintegrasikan dengan lebih mudah dan mendukung tumbuhnya lingkungan virtual di jaringan itu sendiri.

Virtualisasi jaringan memecahkan banyak persoalan jaringan di *data center* jaman sekarang, menolong perusahaan dalam sentralisasi program dan *network provision* secara *on-demand* tanpa harus menyentuh infrastruktur dasar secara fisik. Dengan virtualisasi jaringan perusahaan dapat mempermudah *deploy* servis, skalabilitas, mengatur *workload* dan sumber daya untuk memenuhi kebutuhan komputasi yang terus berevolusi.

2.4 Mininet

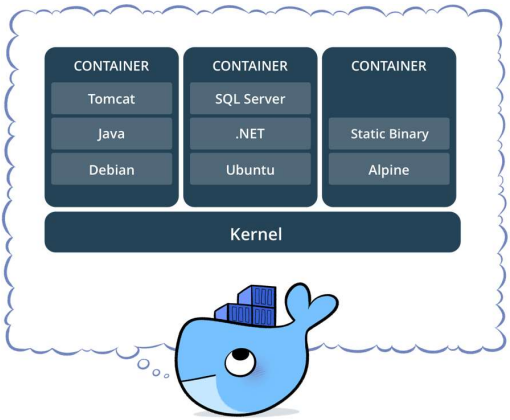
Mininet adalah sebuah *emulator* jaringan yang dapat membuat jaringan yang berisi *Virtual Host, Controller, Switch,* dan *Link*[4]. Mininet berjalan pada platform linux dan sudah didukung oleh switch berbasis OpenFlow untuk fleksibilotas *routing* dan *Software-Defined Networking* (SDN). Mininet digunakan oleh peneliti untuk membangun jaringan virtual untuk keperluan penelitian, pengetesan, maupun *prototyping* jaringan. Mininet berjalan di atas bahasa pemrograman Python.

Mininet menyediakan cara yang mudah untuk mendapatkan *system behavior* yang benar dan untuk bereksperimentasi dengan topologi.

2.5 Docker

Container adalah *image* software yang ringan, mandiri, dan memiliki package yang dapat dijalankan yang sudah memiliki semua komponen yang diperlukan untuk menjalakannya (kode, *runtime, system tool, system libraries,* dan *setting*)[5].

Container maemisahkan software dari sekitarnya, sebagai contoh perbedaan antara pengembangan dan *staging,* juga merupakan solusi atas konflik antar tim yang menggunakan software yang berbeda di infrastruktur yang sama. Gambar 2.2 merupakan ilustrasi bagaimana *container* bekerja.



Gambar 2.2 Ilustrasi dari Container Docker[5]

2.6 Containernet

Containernet merupakan fork dari mininet yang memungkinkan user untuk menggunakan docker sebagai *host* dari mininet[6]. Hal ini memberikan fungsionalitas yang menarik untuk membangun jaringan maupun *cloud.* Integrasi dilakukan dengan melakukan *subclassing* dari mininet *host* asli. Containernet dibangun berdasarkan Mininet versi 2.2.1

Fitur fitur dari Containernet sendiri antara lain adalah:

- Menambah atau menghapus *Container* didalam topologi Mininet.
- Melakukan perintah didalam *Container* dengan menggunakan CLI dari Mininet.
- Menghubungkan *Container* ke topologi.
- Menggubah topologi secara dinamis seperti *cloud.*
- Mengatur limitasi sumber daya dari *Container.*
- Kontrol terhadap trafik *link.*
- Testing unit secara otomatis.
- Installasi secara otomatis lewat Ansible *playbook.*

2.7 Python

Python merupakan salah satu bahasa pemrograman tingkat tinggi yang bersifat interpreter, interaktif serta *object oriented*[7]. Python dapat beroperasi pada banyak *platform* berbeda seperti UNIX, Mac, Windows ataupun yang lainnya.

Python telah menyediakan banyak modul sehingga penggunaannya menjadi sangat mudah. Python memiliki struktur data tingkat tinggi yang efisien. Ada dua versi yang disediakan Python yaitu versi 2 dan versi 3 yang masing-masing versinya memiliki kekurangan dan kelebihan tersendiri.

2.8 SDN Controller (POX)

POX adalah platform software jaringan yang ditulis berdasarkan Bahasa Python. Awalnya POX ditujukan sebagai OpenFlow *controller,* namun sekarang dapat juga digunakan sebagai OpenFlow *switch*, dan berguna untuk membangun software jaringan secara umum[8]. POX memerlukan Python 2.7 untuk dapat berjalan dengan baik dan dapat dijalankan di Linux, Mac OS, dan juga Windows.

2.9 Flask

Flask adalah sebuah microframework untuk Python berbasis Werkzeug, Jinja 2 dan niat baik[9]. Flask berfungsi sebagai pengganti PHP POST dan GET yang dimana pengendali pertukaran data dari HTML ke database. Flask juga menggantikan fungsi Apache sebagai *webserver* dimana flask berjalan di http://localhost:5000/. Gambar 2.3 menunjukkan contoh pengaplikasian Flask.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Gambar 2.3 Contoh pengaplikasian Flask[9]

2.10 HTML5

Hypertext Markup Language revision 5 (HTML5) adalah Bahasa *markup* untuk struktur dan tampilan/presentasi dari konten *World Wide Web* (WWW)[10].

HTML5 mendukung fitur fitur dari HTML tradisional dan XHTML-*style* sintax dan banyak fitur baru, memakai API baru dan juga memiliki fitur *error handling.*

2.11 Bootstrap

Bootstrap adalah sebuah *front-end framework* gratis untuk mempercepat dan mempermudah pengembangan dari *web.* Bootstrap terdiri dari desain tempelate berbasis HTML dan CSS juga beberapa *plugins* JavaScript. Bootstrap juga memudahkan user untuk membuat desain responsive dengan mudah[11].

Terdapat dua versi utama yang ditulis ulang yaitu v2 dan v3. Pada Bootstrap 2, ada tambahan fungsionalitas responsif untuk keseluruhan *framework* sebagai *stylesheet* opsional. Sedangkan Bootstrap 3 yang ditulis ulang memanfaatkan fungsional responsif dengan pendekatan pertama *mobile*.

2.12 MySQL

MySql adalah system managemen database SQL *open-source* terpopuler, yang dimana dikembangkan, distribusikan dan didukung oleh Oracle Corporation

MySQL merupakan turunan salah satu konsep utama dalam basis data yang sudah ada sejak lama, yaitu SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian basis data yang mengutamakan pemilihan atau seleksi dan pemasukan data. MySQL memungkinkan pengoperasian data dapat dikerjakan dengan mudah secara otomastis. MySQL memiliki beberapa kelebihan, antara lain portabilitas, *open-source*, skalabilitas, *multi-user*, keamanan cukup baik dan lain sebagainya.

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN PERANGKAT LUNAK

Bab ini membahas khusus mengenai analisis dan perancangan perangkat lunak yang akan dikembangkan. Secara teknis, aktivitas perancangan merupakan salah satu aktivitas yang sangat penting dilakukan dalam rangka pengembangan perangkat lunak. Beberapa hal yang secara umum dibahas dalam bab ini adalah deskripsi umum sistem, arsitektur umum sistem, diagram kasus penggunaan, perancangan basis data, diagram alur, dan desain antar muka perangkat lunak.

3.1 Deskripsi Umum Sistem

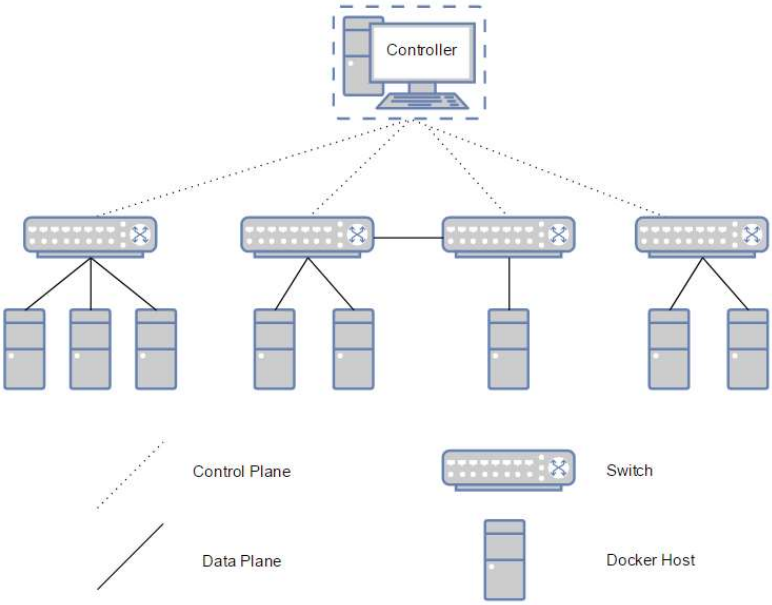
Pada Tugas Akhir ini akan dibangun sistem Virtual Data Center diatas platform Linux Server 14.04.5 LTS dengan Mininet sebagai virtualisasi jaringan dan beragam macam image Docker sebagai *host* dari sistem ini. Sistem Virtual Data Center ini dibangun berdasarkan teknologi arsitektur jaringan terbaru yaitu Software Defined Network (SDN) dengan POX sebagai controller-nya. Untuk mempermudah akses dan penggunaan disediakan web UI berbasis HTML 5 dan Bootstrap untuk frontend, Flask untuk backend dan MySQL untuk database.

Web UI memliki fungsi yang memudahkan para user untuk mengelola *Virtual Data Center*-nya masing masing. Fungsi – fungsi yang ada pada website meliputi pembuatan user baru, menampilkan Docker *host*, pengelolaan Docker (*Up, Down, Delete),* penambahan Docker, dan membuat koneksi antar *user/switch.*

3.2 Arsitektur Umum Sistem

Rancangan *Virtual Data Center* dari sistem dapat dilihat pada Gambar 3.1. Berdasarkan perancangan arsitektur sistem pada Gambar 3.1, menunjukkan sebuah arsitektur *Virtual Data Center*

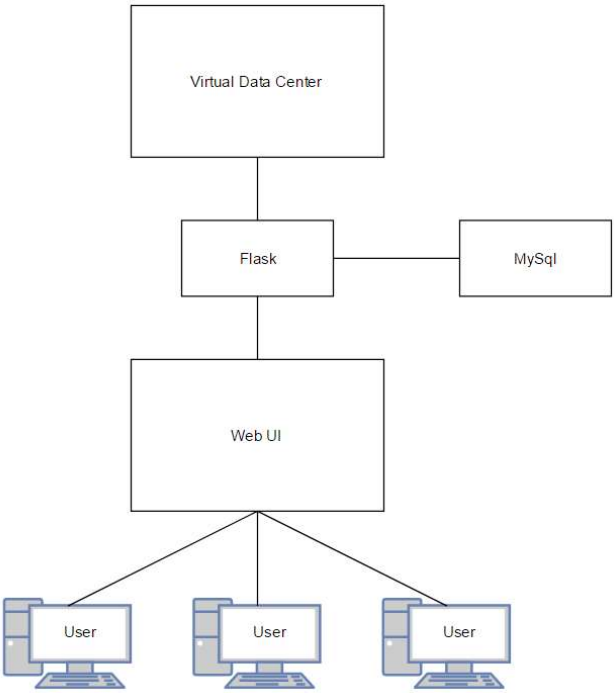
berbasis *Container* sederhana yang menggunakan *SDN Controller*. Terlihat terjadinya pemisahan antara *data plane* dan *control plane*. *SDN Controller* memiliki koneksi langsung berupa *control plane* dengan masing-masing *virtual switch*, sendangkan masing-masing *virtual switch* memiliki koneksi *data plane* dengan *virtual switch* lainnya yang digunakan untuk mengirim paket.



Gambar 3.1 Arsitektur Umum *Virtual Data Center*

Setiap switch di arsitektur pada Gambar 3.1 bersifat independen dimana setiap switch tidak dapat berkomunikasi satu sama lain kecuali dibuat jalur *data plane* antar switch yang ingin disambungkan. Hal ini dilakukan supaya *data plane* antar user tidak bocor satu sama lain, sehingga menjamin keamanan data antar user. Pada saat mendaftar user otomatis diberikan satu switch di dalam *Virtual Data Center* ini. Untuk menjawab permasalahan

agilitas *cloud,* penulis memberikan user pilihan untuk menyambungkan *data plane* kepada switch user lain.



Gambar 3.2 Arsitektur Umum Sistem

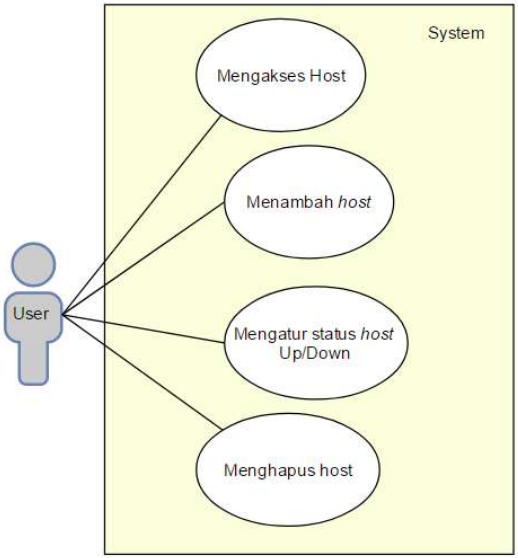
Gambar 3.2 menunjukkan arsitektur umum dari sistem secara keseluruhan. Untuk memudahkan user mengakses *Virtual Data Center* penulis membuat UI berupa web berbasis Flask dan MySql. Flask dipilih karena berjalan diatas bahasa pemrograman Python sehingga lebih mudah diintegrasikan untuk mengontrol *Virtual Data Center.* Flask juga mengatur keluar masuknya data ke MySql untuk menyimpan inrformasi dan data dari user. Flask juga mengolah data untuk ditampilkan ke web untuk mempermudah user dalam memahami data tersebut.

3.3 Perancangan Website (Server)

Website yang akan dibangun merupakan sebuah website untuk memonitor atau memantau dan mengontrol switch dan *host* Docker di dalam *Virtual Data Center*. *Host – host* yang telah dibuat dan terdaftar di dalam database akan ditampilkan pada web berupa tabel. Di dalam tabel tersebut akan ada infromasi terkait dengan *host* tersebut seperti status dan alamat dari *host*.

Kegunaan lain yang dimiliki website ada beberapa hal. Beberapa hal tersebut meliputi, user *login,* mendaftarkan user baru, menampilkan alamat *switch*, mengkoneksikan *switch* dengan *switch* user lain, menambahkan *host* baru, menampilkan *host* dengan informasinya, merubah status dari *host* (*UP/*DOWN) serta dapat menghapus *host* yang telah terdaftar.

3.3.1 Perancangan Diagram Kasus Penggunaan



Gambar 3.3 Diagram Kasus Penggunaan

Pada bagian ini akan dijelaskan rincian kasus penggunaan (*use case*) untuk pengguna. Kasus penggunaan yang akan dibuat terdiri dari beberapa hal yaitu mengakses *host,* menambah *host,* mengatur status *host,* dan menghapus *host*. Gambar 3.3 merupakan gambar diagram kasus penggunaan (*use case*). Berdasarkan diagram kasus penggunaan yang telah dibuat, berikut ini rincian dari setiap kasus penggunaan yang ada :

3.3.1.1 Mengakses Docker *Host*

Untuk dapat mengakses Docker *host,* user diharuskan melakukan SSH ke server *Virtual Data Center.* Setelah melakukan SSH user dipersilahkan melakukan *excec* ke Docker sesuai dengan id Docker yang tertera didalam website. Setelah masuk ke dalam Docker user dapat melakukan konfigurasi sesuai penggunaannya nanti.

3.3.1.2 Menambah *Host*

Kasus penggunaan ini merupakan halaman untuk menambah *host* yang terkoneksi dengan *switch* masing masing user. Untuk menambahkan *host* user hanya perlu memasukkan tiga inputan kedalam form yang telah disediakan di dalam website. Tiga inputan tersebut adalah nama Docker *host.* Jenis dari Docker *image,* yang dimana penulis telah menyediakan macam macam *image* antara lain, MySql, PHPMyAdmin, Ubuntu. Inputan yang ketiga berupa inputan opsional yaitu setting tambahan untuk Docker seperti CPU, Memory *limitation* dan *volume.* Setelah memencet tombol tambah *host.* Docker *host* secara otomatis akan dibuat, tersambung pada *switch* dan siap untuk digunakan.

3.3.1.3 Mengatur Status *Up/Down* Pada *Host*

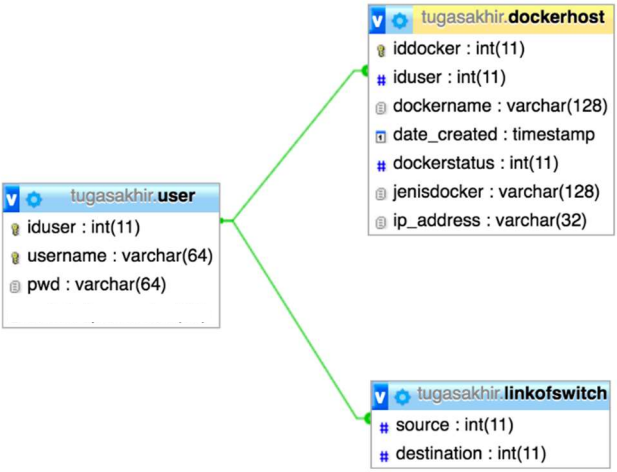
Kasus penggunaan ini berada di tabel yang menampilkan daftar Docker *host* yang berada di halaman utama untuk memudahkan user dalam mengatur dan memonitor Docker *host*

miliknya. User dapat dengan mudah mengubah status / menghentikan dan menyalakan Docker *host* dengan memencet tombol yang ada didalam tabel. Dengan otomatis Docker *host* akan berhenti datau berjalan sesuai dengan status yang diinginkan oleh user.

3.3.1.4 Menghapus *Host.*

Sama seperti pengaturan status, kasus penggunaan ini pun ditaruh di tabel daftar Docker *host* yang berada di halaman utama untuk memudahkan user dalam mengatur dan mengelola Docker *host* miliknya. User dapat dengan mudah menghapus Docker *host* yang diinginkan hanya dengan memencet tombol yang telah disediakan didalam list tabel daftar Docker *host.*

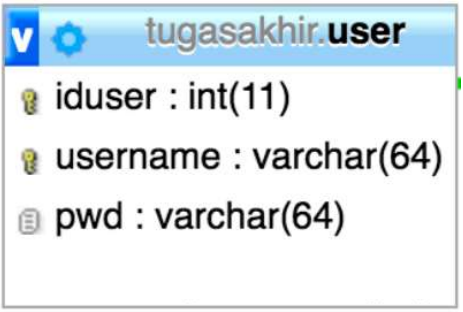
3.3.2 Perancangan Basis Data (Website)



Gambar 3.4 Skema Basis Data (Website)

Perancangan basis data merupakan satu tahap untuk merancang basis data yang akan digunakan untuk menampung data – data yang diperlukan. Basis data inilah yang nantinya akan diolah dan ditampilkan dalam website. Sistem manajemen basis data yang akan digunakan ialah MySQL. Beberapa hal yang akan disimpan dalam basis data adalah data – data mengenai user dan data – data dari Docker *host*. Gambar 3.4 merupakan rancangan skema dari basis data. Berikut ini penjelasan lebih rinci mengenai setiap entitas dan atribut dalam basis data.

3.3.2.1 Tabel *users*



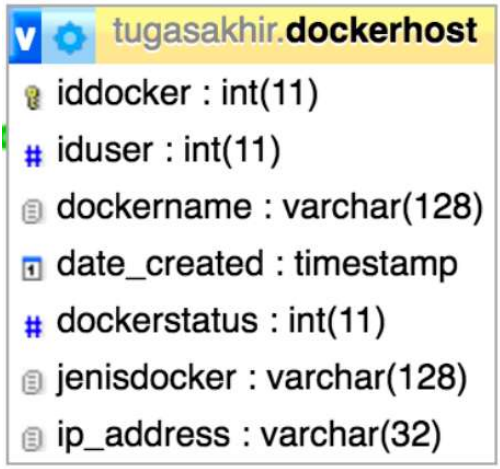
Gambar 3.5 Skema Tabel *user*

Gambar 3.5 adalah skema dari tabel *user*. Fungsi dari tabel (entity) ini adalah untuk menyimpan data – data dari Pengguna. Data Pengguna yang disimpan di dalamnya meliputi *iduser*, *username*, *password* dan *switch_ip*. Tabel 3.1 merupakan tabel yang menjelaskan detail dari tabel *user*.

Tabel 3.1 Detail Tabel *user*

No	Nama Atribut	Tipe Data	Keterangan
1.	*Iduser*	*Integer*	Merupakan *primary key* dari tabel *user*. Diatur sebagai *auto increment*
2.	*Username*	*varchar(64)*	*username* dari Pengguna yang digunakan sebagai pengenal untuk masuk (login) ke dalam website
3.	*Password*	*varchar(64)*	*password* dari Pengguna yang digunakan sebagai kode pengaman untuk masuk (login) ke dalam website

3.3.2.2 Tabel *dockerhost*



Gambar 3.6 Skema Tabel *dockerhost*

Gambar 3.6 adalah skema dari tabel *dockerhost*. Fungsi dari tabel (entity) ini adalah untuk menyimpan data – data mengenai Docker *host*. Data yang disimpan di dalamnya meliputi *iddocker*, *iduser*, *dockername*, *date_created*, *dockerstatus*, *jenisdocker,* serta

ip_address. Data ini didapatkan ketika proses penambahan Docker *host*. Sedangkan data status (*dockerstatus*) diperbarui secara berkala sesuai dengan status dari Docker tersebut. Tabel 3.2 merupakan tabel yang menjelaskan detail dari tabel *dockerhost*.

Tabel 3.2 Detail Tabel *dockerhost*

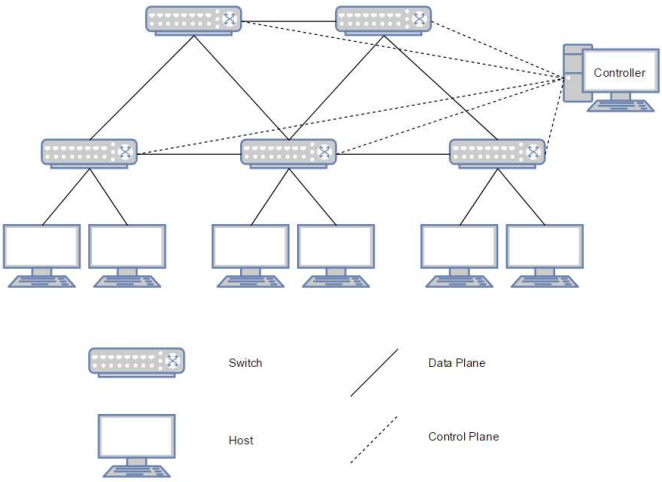
No	Nama Atribut	Tipe Data	Keterangan
1.	*Iddocker*	*Integer*	Merupakan *primary key* dari tabel *dockerhost*. Diatur sebagai *auto increment*
2.	*Iduser*	*Integer*	Merupakan *foreign key* yang terhubung dengan tabel *user*. Satu *user* dapat memiliki banyak *dockerhost* (*one to many*).
3.	*Dockername*	*varchar(128)*	Nama dari Docker *host* yang didaftarkan
4.	*date_created*	*Timestamp*	Tanggal dan waktu Docker *host* ditambahkan.
5.	*Dockerstatus*	*Integer*	Merupakan penunjuk status dari Docker *host* tersebut. 1 untuk menyala dan 0 untuk mati.
6.	*Jenisdocker*	*varchar(128)*	Menunjukkan jenis *image* yang digunakan oleh Docker *hsot*.
7.	*ip_address*	*varchar(32)*	Menunjukkan alamat IP dari Docker *hsot* yang dimaksud.

3.4 Perancangan Sistem Jaringan *Virtual Data Center*

Pada bagian ini akan dilakukan perancangan dari *Virtual Data Center*. *Virtual Data Center* akan dibangun dengan Containernet sebagai dasarnya. Dimana Containernet sendiri merupakan modifikasi dari Mininet. Mininet merupakan emulator untuk mensimulasikan jaringan yang besar di dalam satu alat.

Mininet dapat digunakan untuk membuat jaringan virtual nyata yang menjalankan *kernel, switch,* dan kode aplikasi *software* didalam sebuah komputer personal dengan cara yang relative mudah. Mininet memungkinkan penulis untuk membuat, berinteraksi, dan mengkostumisasi sebuah *prototype* SDN untuk mensimulasikan topologi jaringan yang mengunakan *switch* OpenFlow. Sedangkan Containernet sendiri mengubah mininet supaya dapat menjalankan topologi jaringan dengan Docker berjalan diatas *host* mininet.

Pada subbab ini, akan dibahas topologi dan konfigurasi dari jaringan. Perancangan topologi dan konfigurasi jaringan secara keseluruhan dilakukan agar dapat lebih mudah memahami cara kerja dari jaringan virtual itu sendiri. Gambar 3.7 Topologi dasar sistem *Virtual Data Center* menunjukkan topologi yang gunakan untuk penulis sebagai topologi dasar dari *Virtual Data Center.*



Gambar 3.7 Topologi dasar sistem *Virtual Data Center*

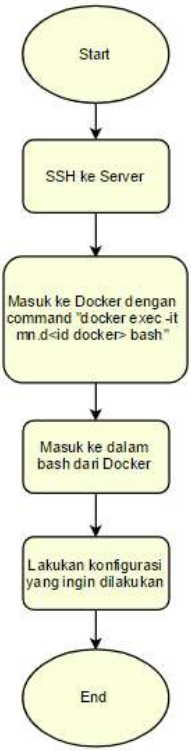
Secara garis besar, sistem jaringan yang berjalan membutuhkan tiga komponen yaitu *switch, host,* dan yang paling penting yaitu *controller.* Ketiga komponen tersebut kemudian disambungkan untuk membentuk sebuah jaringan virtual berbasis SDN yang berkerja dengan baik.

Switch dari topologi tersebut merupakan OpenFlow 1.0 dengan POX sebagai *controller*-nya. Penulis menggunakan topologi dengan dengan banyak sekali jalur yang redundan untuk mempermudah percobaan pada POX *controller.* Percobaan yang dilakukan yaitu mengubah jalur koneksi jaringan dengan menonaktifkan jalur yang telah ada sehingga memaksa POX *controller* untuk beradaptasi dan mencari jalur baru untuk topologi ini. Dikatakan berhasil jika *controller* membuat jalur baru yang berbeda. Setelah berhasil maka konfigurasi dalam jaringan dapat digunakan kedalam sistem yang kemudian diimplementasikan ke dalam web UI.

3.5 Perancangan Sistem

Pada bagian ini akan dilakukan perancangan dari sistem secara keseluruhan. Di dalam sub-bab ini dijelaskan setiap alur dari *use case* / diagram alur yang ada didalam sistem ini. *Use case* tersebut antara lain adalah mengakses *host,* menambahkan *host,* mengatur status dari *host,* dan menghapus *host.*

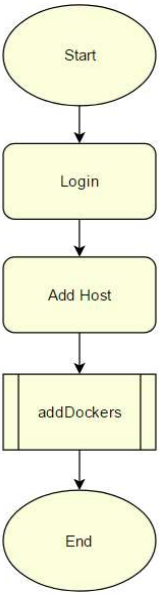
3.5.1 Perancangan Diagram Alir (Mengakses *Host*)



Gambar 3.8 Diagram Alir Sistem (Mengakses *host*)

Untuk dapat masuk ke dalam Docker *host* yang dimiliki user terlebih dahulu harus melakukan SSH ke dalam *Virtual Data Center*. Setelah itu user harus menuliskan *command* "docker exec -it mn.d<nomor id_docker> bash". Dimana nomor id Docker dapat dilihat / didapatkan ketika user masuk dan melakukan login ke website. Nomor id terletak di tabel list Docker *host* beserta informasi lainnya. Gambar 3.8 Diagram Alir Sistem (Mengakses *host*) merupakan gambaran umum jalannya proses menampilkan *host* secara keseluruhan dalam bentuk diagram alir.

3.5.2 Perancangan Diagram Alir (Menambahkan *Host*)



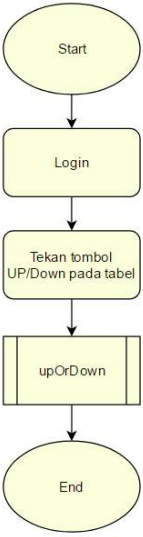
Gambar 3.9 Diagram Alir Sistem (Menambahkan *Host*)

Untuk dapat menambahkan *host* user terlebih dahulu harus melakukan login yang dimana langsung dilempar ke halaman utama. User kemudian harus pergi ke halaman *Add Host* dan mengisi form yang telah disediakan. Gambar 3.9 merupakan gambaran umum jalannya proses menambahkan koneksi *switch* secara keseluruhan dalam bentuk diagram alir.

Pada bagian subproses *tambahHost*, Flask akan mengirimkan perintah untuk menambahkan data *host* ke MySql. Kemudian Flask akan memerintahkan *Virtual Data Center* membuat Docker *host* beserta link ke *switch* milik user. Gambar 3.10 merupakan diagram alir subproses *tambahHost*.

Gambar 3.10 Diagram Alir Subproses *tambahLink*

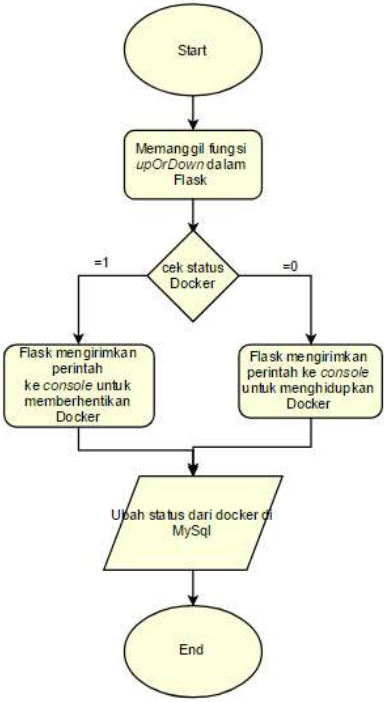
3.5.3 Perancangan Diagram Alir (Mengatur Status *Host*)



Gambar 3.11 Diagram Alir Sistem (Mengatur Status *Host*)

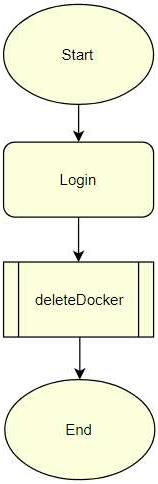
Untuk dapat mengatur status *host* user terlebih dahulu harus melakukan login yang dimana langsung dilempar ke halaman utama. User kemudian harus menekan tombol UP/Down pada tabel list *host*. Gambar 3.11 merupakan gambaran umum jalannya proses mengatur status *host* secara keseluruhan dalam bentuk diagram alir.

Pada bagian subproses *upOrDown*, Flask akan mengirimkan perintah untuk mengubah data *host* sesuai status ke MySql. Kemudian Flask akan menjalankan perintah ke *console* lewat POPEN untuk mengubah status dari Docker yang berjalan. Gambar 3.12 merupakan diagram alir subproses *upOrDown*.



Gambar 3.12 Diagram Alir Subproses *upOrDown*

3.5.4 Perancangan Diagram Alir (Menghapus *Host*)



Gambar 3.13 Diagram Alir Sistem (Menghapus *Host*)

Untuk dapat menghapus *host* user terlebih dahulu harus melakukan login yang dimana langsung dilempar ke halaman utama. User kemudian harus menekan tombol Delete pada tabel list *host* Gambar 3.13 merupakan gambaran umum jalannya proses mengatur status *host* secara keseluruhan dalam bentuk diagram alir.

Pada bagian subproses *deleteDocker*, Flask akan menjalankan perintah ke *console* untuk menghapus Docker yang diinginkan. Kemudian data di MySql diupdate untuk menghapus data Docker yang bersangkutan Gambar 3.14 merupakan diagram alir subproses *deleteDocker*.

Gambar 3.14 Diagram Alir Subproses *deleteDocker*

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini berisi penjelasan mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa *Pseudocode* untuk membangun program.

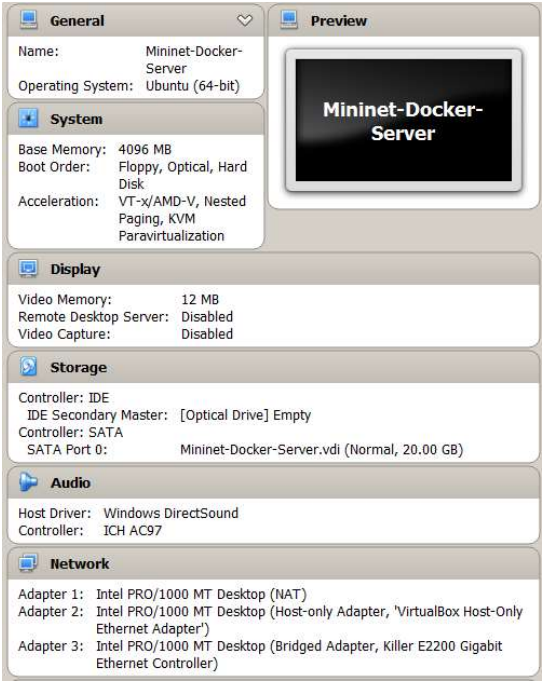
4.1 Lingkungan Implementasi

Implementasi *Virtual Data Center* menggunakan linux *container* berbasis Docker dan SDN menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i7-5700 HQ @ 2.7 GHz (8 CPUs)
	Memori	8 GB 1600 MHz DDR3 X 2
Perangkat Lunak	Sistem Operasi	Windows 10
	Perangkat Pengembang	Oracle VM Virtual Box v. 5.1.16 r113841. Xterm dan Putty.

Sedangkan untuk spesifikasi dari Virtual Box ditunjukkan pada Gambar 4.1. Virtual Box menjalankan Ubuntu server 14.04 LTS dengan versi 64 –bit. Virtual Box berjalan dengan konfigurasi 4 GB RAM, 1 CPU dengan *caps* 100%, 20 GB HardDrive. Selain itu Virtual Box memiliki tiga adapter jaringan. Jaringan dengan jenis NAT untuk meneruskan koneksi internet dari PC induk, *Host only adapter* untuk koneksi SSH, dan *Bridged adapter* untuk meneruskan koneksi LAN PC induk.



Gambar 4.1 Spesifikasi dari Virtual Box

4.2 Implementasi *Virtual Data Center* (Pembuatan Docker *Host Machine*)

Pada subbab implementasi *Virtual Data Center* ini menjelaskan tentang pembangunan perangkat lunak secara detail dan menampilkan *pseudocode* implementasi dari tiap kasus penggunaan pada bagian perancangan dan fungsi yang dibuat untuk menunjang tercapainya implementasi tersebut. Disini dijelaskan tentang logika dari penambahan Docker *host* kedalam virtualisasi Containernet.

Fungsi ini bertujuan untuk membuat Docker *host machine* baru. Untuk menambahkan Docker *host machine* baru diperlukan nama untuk Docker *machine*, jenis dari Docker *machine* dan

konfigurasi tambahan seperti *cpu_quota* jika diperlukan. Setelah itu Docker *host machine* perlu disambungkan ke *switch* milik user yang bersangkutan. *Pseudocode* dari fungsi ini dapat dilihat di Pseudocode 4.1:

```
1  FUNCTION tambahDocker(self, nomor, ipAddress,
2  jenis, switchNo):
3       net #declare controller
4       nomor <- 'd'+ nomor
5       d <- net.addDocker(nomor,defaultRoute=None,
   dimage=jenis)
6       net.addLink(d,net.get(switchNo),params1{"ip"
   :ipAddress})
7  ENDFUNCTION
```

Pseudocode 4.1 Membuat Docker *Host Machine Baru*

4.3 Implementasi Website (Flask)

Pada subbab implementasi website ini menjelaskan tentang pembangunan perangkat lunak secara detail dan menampilkan *pseudocode* implementasi dari tiap kasus penggunaan pada bagian perancangan dan fungsi yang dibuat untuk menunjang tercapainya implementasi tersebut. Disini dijelaskan tentang logika dari pengaturan Docker *host* melalui Flask sehingga user dapat mengatur dan mendapatkan informasi Docker *host* dari web antarmuka yang telah disediakan.

4.3.1 Membuat Docker *Host* Baru

Fungsi ini bertujuan untuk membuat Docker *host* baru. Untuk menambahkan Docker *host* baru diperlukan hasil inuputan *form* dari user. Inputan dari user tersebut berupa nama Docker dan juga jenis *image* dari Docker. Akhir dari fungsi ini adalah memanggil fungsi tambahDocker() di *script* Containernet. Selain itu dibagian ini *script* melakukan update ke database ketika ada Docker *host* baru yang dibuat. *Pseudocode* dari fungsi ini dapat dilihat di Pseudocode 4.2.

```
1  FUNCTION addDockers():
```

```
2    IF request.method = 'POST':
3      username_form <- session['username']
4      dockerName_form<-request.form['dockerName']
5      dockerImage_form<-request.form['dockerImage']
6      hasil<-idDocker terakhir dari database
7      idDocker_form=hasil+1
8      hasil<-idUser dari database
9      idUser_form=hasil
10     ipAddress="10.0.0."+idDocker_form
11     Tambah data Docker ke Database
12     tambahDocker(idDocker_form,ipAddress+"/8"
13   dockerImage_form,"s"+idUser_form)
       RETURN redirect()
14   ENDFUNCTION
15
```

Pseudocode 4.2 Membuat Docker *Host* Baru

4.3.2 Mengatur Status *Up* dan *Down* Docker *Host*

Fungsi ini bertujuan untuk mengubah status *Up* dan *Down* pada Docker *host*. Docker *host* masuk dalam keadaan *Up* ketika Docker dalam keadaan menyala dan siap untuk digunakan. Sedangkan Docker *host* dikatakan dalam keadaan *Down* ketika Docker berada dalam keadaan diberhentikan/stop. Sebelum melakukan perubahan *state/*status dari Docker dilakukan pengecekan ke database mengenai status dari Docker yang bersangkutan. Ketika status dari Docker dalam keadaan menyala atau di *script* ini berarti 1 dan fungsi dipanggil secara otomatis *script* akan melakukan pemberhentian dari Docker yang bersangkutan dan juga mengupdate data status dari Docker di database menjadi 0. Begitu juga sebaliknya, jika fungsi dipanggil dalam keadaan Docker yang bersangkutan mati/berhenti atau 0 maka fungsi akan menyalakan Docker yang bersangkutan dan mengubah data status di database menjadi 1. Fungsi ini dipanggil dengan memencet tombol di tabel Docker *list.* *Pseudocode* dari fungsi pengaturan status Docker dapat dilihat di Pseudocode 4.3.

```
1    FUNCTION addDockers():
2     IF request.method = 'POST':
```

```
3     dockerID_form <- request.form['dockerID']
4     mndocker <- "mn.d"+dockerID_form
5     hasil <- dockerstatus dari database
6     dockerStatus_form <- str(hasil)
7     IF dockerStatus_form = '1':
8       Update dockerstatus ke '0'
9       Output = Popen(['docker','stop',mndocker],
10  stdout=PIPE)
11      Output.communicate()
12    ELSE:
13      Update dockerstatus ke '1'
14      Output = Popen(['docker','start',mndocker],
    stdout=PIPE)
15      Output.communicate()
16    RETURN redirect()
17  ENDFUNCTION
```

Pseudocode 4.3 Mengatur Status Docker *Host*

4.3.3 Menghapus Docker *Host*

Fungsi ini bertujuan untuk menghapus Docker yang telah ada didalam sistem. Untuk dapat menghapus Docker yang telah ada user dapat memanggil fungsi ini yaitu lewat tombol di daftar tabel Docker di dalam website untuk Docker yang bersangkutan. Sebelum fungsi ini menghapus Docker, fungsi mengecek dockerstatus di dalam database. Jika dockerstatus bernilai '0' atau dapat dibilang Docker dalam keadaan mati/berhenti maka Docker dapat langsung dihapus. Sebaliknya jika dockerstatus di dalam database bernilai '1' yang dimana berarti Docker masih dalam keadaan menyala, Docker harus dimatikan terlebih dahulu sebelum dapat dihapus dari sistem. Setelah menghapus Docker fungsi memperbaharui Database dengan menghapus data Docker dari database untuk Docker yang bersangkutan. *Pseudocode* dari fungsi ini dapat dilihat di Pseudocode 4.4.

```
1   FUNCTION deleteDocker():
2    IF request.method = 'POST':
3      dockerID_form <- request.form['dockerID']
4      mndocker <- "mn.d"+dockerID_form
```

```
5       hasil <- dockerstatus dari database
6       dockerStatus_form <- str(hasil)
7       IF dockerStatus_form = '1':
8         Update dockerstatus ke '0'
9         Output = Popen(['docker','stop',mndocker],
    stdout=PIPE)
10        Output.communicate()
11      Output = Popen(['docker','rm',mndocker],
    stdout=PIPE)
12      Output.communicate()
13      Delete data Docker di Database
14      RETURN redirect()
15  ENDFUNCTION
```

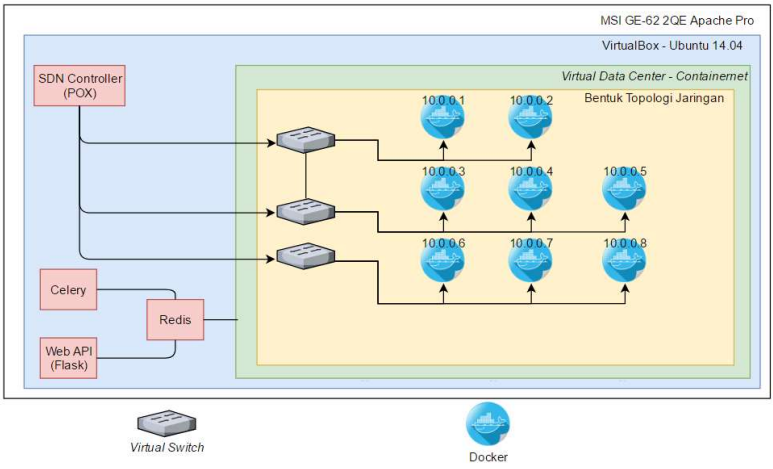
Pseudocode 4.4 Menghapus Docker *Host*

BAB V
HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi pada implementasi Docker *machine* dalam *Virtual Data Center* berbasis SDN. Hasil uji coba didapatkan dari implementasi bab 4 dengan skenario pengujian. Bab ini berisikan pembahasan mengenai lingkungan pengujian, skenario pengujian, dan hasil pengujian.

5.1 Lingkungan Pengujian

Lingkungan pengujian mencakup perangkat keras maupun perangkat lunak yang digunakan untuk melakukan uji coba. Uji coba dilakukan pada sebuah mesin virtual (Virtual Box) yang berjalan diatas perangkat fisik *laptop*. Lingkungan uji coba dapat dilihat di Gambar 5.1.



Gambar 5.1 Bagan Lingkungan Pengujian

- Laptop MSI GE-62 2QE Apache Pro
    - Intel(R) Core(TM) i7-5700HQ @ 2.70 GHz (4 CPUs)
    - RAM 8 GB 1600 MHz DDR3 x 2
    - Windows 10 Home Single Language 64-bit (10.0, Build 14393)
- Mesin *Virtual* Ubuntu 14.04
    - Ubuntu 14.04 64bit
    - RAM 4 GB
    - 20GB of HardDrive
    - Single CPU Processor
- Mesin *Virtual* Ubuntu 14.04
    - Ubuntu 14.04 64bit
    - RAM 8 GB
    - 20GB of HardDrive
    - 4 CPU Processors

5.2 Skenario Pengujian Performa

Diperlukan beberapa skenario untuk menguji sistem yang telah dibuat. Skenario pengujian performa diperlukan untuk menguji kinerja dari sistem yang telah dibuat. Hal yang akan diuji adalah kecepatan dan kehandalan dari setiap Docker *machine container* dari masing masing skenario pengujian.

5.2.1 Skenario Pengujian (SP-01) – Stress Test Jumlah Docker *Machine* (4GB RAM, 1 Core CPU)

Skenario pengujian ini menguji kinerja dan ketahanan dari sistem virtual di Virtual Box. Pengujian ini dilakukan untuk menguji seberapa banyak Docker *machine* yang dapat dijalankan dengan spesifikasi lingkungan dengan 4GB RAM dan 1 Core CPU. Pengujian dilakukan dengan Docker *image* dengan kebutuhan daya terbesar yaitu Ubuntu:Trusty. Skenario pengujian dapat dilihat di Tabel 5.1.

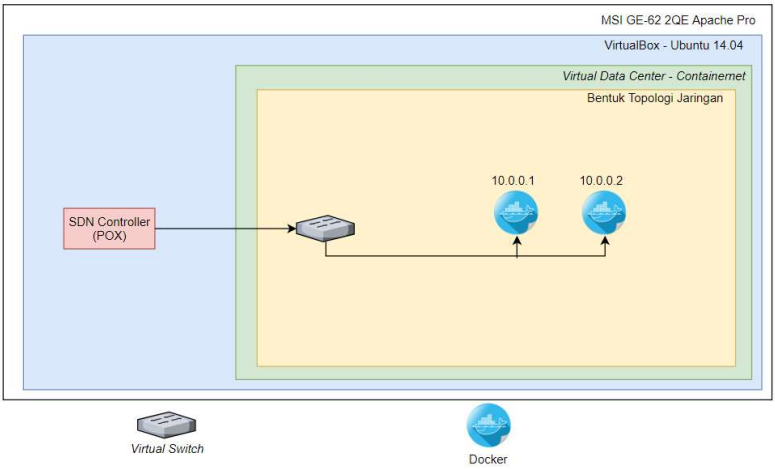
Tabel 5.1 Skenario Pengujian (SP-01)

ID	**SP-01**
Nama	Uji Coba Stress Test Jumlah Docker *Machine*
Tujuan Uji Coba	Menguji seberapa banyak Docker *Machine* yang dapat berjalan di spesifikasi lingkungan percobaan yang tertera
Kondisi Awal	Tidak ada Docker yang berjalan, aktif maupun pasif, pada sistem
Metode Pengujian	Uji *strees test* terhadap jumalah Docker menggunakan script yang telah dibuat
Lama Pengujian	-
Skenario	1. Pastikan tidak ada Docker yang berjalan aktif maupun pasif dengan menjalankan *script clear_crash.sh* 2. Jalankan *script* yang telah dibuat untuk menjalankan Docker dengan jumlah yang tinggi. 3. Tunggu hingga sistem lingkungan menjadi lamban / tidak responsive atau *script* terjadi error dan tidak dapat menambah jumlah Docker yang berjalan. 4. Jalankan "docker ps –a" untuk mengetahui jumlah Docker yang berjalan
Masukan	-
Keluaran	Banyaknya Docker yang dapat dijalankan pada lingkunan dengan spesifikasi yang tertera.
Hasil yang Diharapkan	-

5.2.2 Skenario Pengujian (SP-02) – Performa TCP antar 2 Docker *Machine* yang Terhubung di 1 *Switch* (4GB RAM, 1 Core CPU)

Skenario pengujian ini menguji kinerja dari Docker *machine* yang saling tersambung di dalam sistem *Virtual Data Center*. Pengujian ini dilakukan untuk menguji kecepatan transfer

data dengan protokol TCP antar Docker *machine*. Pengujian dilakukan dengan Docker *image* dengan *library networking* yang sudah merupakan fitur dari *image* Docker tersebut yaitu Ubuntu:Trusty. Skenario pengujian dapat dilihat di Gambar 5.2 dan Tabel 5.2.



Gambar 5.2 Skenario Pengujian (SP-02)

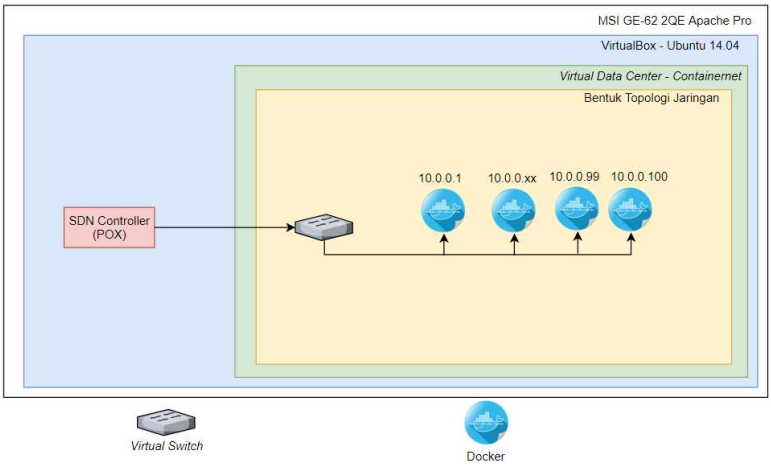
Tabel 5.2 Skenario Pengujian (SP-02)

ID	**SP-02**
Nama	Uji Coba Performa Transfer Data antar 2 Docker *Machine* Yang Terhubung di 1 *Switch*
Tujuan Uji Coba	Menguji kinerja transfer data dengan protokol TCP
Kondisi Awal	Terdapat 2 buah Docker *machine* yang terhubung ke dalam 1 switch
Banyak Docker *machine*	2
Metode Pengujian	Uji kecepatan data transfer menggunakan Iperf
Lama Pengujian	15 Detik

Skenario	1. Sambungkan Docker *host* pertama ke switch 2. Sambungkan Docker *host* kedua ke 3. Buat *Iperf server* pada Docker *host* pertama 4. Lakukan *Iperf client* pada Docker *host* kedua
Masukan	-
Keluaran	Kecepatan rata rata dari jaringan
Hasil yang Diharapkan	-
Penggunaan Memori (RAM)	260 MB
Penggunaan CPU	1.4% - 3.4%

5.2.3 Skenario Pengujian (SP-03) – Performa TCP antar 2 Docker *Machine* dengan 100 Docker *Machine* Terhubung di 1 Switch (4GB RAM, 1 Core CPU)

Skenario pengujian ini menguji kinerja dari Docker *machine* yang saling tersambung di dalam sistem *Virtual Data Center*. Pengujian ini dilakukan untuk menguji kecepatan transfer data dengan protokol TCP antar Docker *machine* dan juga apakah jumlah dari Docker *machine* yang berjalan dan terhubung pada satu *switch* mempengaruhi kecepatan transfer data antar 2 Docker *machine*. Pengujian dilakukan dengan Docker *image* dengan *library networking* yang sudah merupakan fitur dari *image* Docker tersebut yaitu Ubuntu:Trusty menggunakan *library Iperf*[12] sebagai alat testing. Skenario pengujian dapat dilihat di Gambar 5.3 dan Tabel 5.3.



Gambar 5.3 Skenario Pengujian (SP-03)

Tabel 5.3 Skenario Pengujian (SP-03)

ID	**SP-03**
Nama	Uji Coba Performa Transfer Data Antar 2 Docker *Machine* dari 100 Docker *machine* yan
Tujuan Uji Coba	Menguji kinerja transfer data dengan protokol TCP
Kondisi Awal	Terdapat 100 buah Docker *machine* yang terhubung ke dalam 1 switch
Banyak Docker *machine*	100
Metode Pengujian	Uji kecepatan data transfer menggunakan Iperf
Lama Pengujian	15 Detik
Skenario	1. Sambungkan Docker *host* pertama sampai ke seratus ke *switch* 2. Buat *Iperf server* pada Docker *host* pertama 3. Lakukan *Iperf client* pada Docker *host* keseratus
Masukan	-

Keluaran	Kecepatan rata rata dari jaringan
Hasil yang Diharapkan	-
Penggunaan Memori (RAM)	3800 MB
Penggunaan CPU	100%

5.2.4 Skenario Pengujian (SP-04) – Stress Test Jumlah Docker *Machine* (8GB RAM, 4 Core CPU)

Skenario pengujian ini untuk mengetahui perbandingan performa dengan spesifikasi yang berbeda. Skenario pengujian dapat dilihat pada Tabel 5.4.

Tabel 5.4 Skenario Pengujian (SP-04)

ID	**SP-04**
Nama	Uji Coba Stress Test Jumlah Docker *Machine*
Tujuan Uji Coba	Perbandingan performa dengan spesifikasi yang berbeda
Kondisi Awal	Tidak ada Docker yang berjalan, aktif maupun pasif, pada sistem
Metode Pengujian	Uji *strees test* terhadap jumalah Docker menggunakan script yang telah dibuat
Lama Pengujian	-
Skenario	5. Pastikan tidak ada Docker yang berjalan aktif maupun pasif dengan menjalankan *script clear_crash.sh* 6. Jalankan *script* yang telah dibuat untuk menjalankan Docker dengan jumlah yang tinggi (200 <) Docker. 7. Tunggu hingga sistem lingkungan menjadi lamban / tidak responsive atau *script* terjadi error dan tidak dapat menambah jumlah Docker yang berjalan.

	8. Jalankan “docker ps –a” untuk mengetahui jumlah Docker yang berjalan
Masukan	-
Keluaran	Banyaknya Docker yang dapat dijalankan pada lingkunan dengan spesifikasi yang tertera.
Hasil yang Diharapkan	Dengan spesifikasi lebih tinggi, jumlah docker yang berjalan akan semakin banyak juga.

5.2.5 Skenario Pengujian (SP-05) – Performa TCP antar 2 Docker *Machine* yang Terhubung di 1 *Switch* (8GB RAM dan 4 Core CPU)

Skenario pengujian ini untuk mengetahui perbandingan performa dengan spesifikasi yang berbeda. Skenario pengujian dapat dilihat pada Tabel Tabel 5.5.

Tabel 5.5 Skenario Pengujian (SP-05)

ID	**SP-05**
Nama	Uji Coba Performa Transfer Data antar 2 Docker *Machine* Yang Terhubung di 1 *Switch*
Tujuan Uji Coba	Perbandingan performa dengan spesifikasi yang berbeda
Kondisi Awal	Terdapat 2 buah Docker *machine* yang terhubung ke dalam 1 switch
Banyak Docker *machine*	2
Metode Pengujian	Uji kecepatan data transfer menggunakan Iperf
Lama Pengujian	15 Detik
Skenario	5. Sambungkan Docker *host* pertama ke switch 6. Sambungkan Docker *host* kedua ke 7. Buat *Iperf server* pada Docker *host* pertama 8. Lakukan *Iperf client* pada Docker *host* kedua
Masukan	-
Keluaran	Kecepatan rata rata dari jaringan

Hasil yang Diharapkan	Kecepatan transfer data akan semakin tinggi
Penggunaan Memori (RAM)	369 MB
Penggunaan CPU	CPU 1 : 0 ~ 1.3% CPU 2: 0 ~ 0.7 % CPU 3 : 0 ~ 2.3 % CPU 4 : 0 ~ 0.7%

5.2.6 Skenario Pengujian (SP-06) – Performa TCP antar 2 Docker *Machine* dengan 100 Docker *Machine* Terhubung di 1 Switch (8GB RAM, 4 Core CPU)

Skenario pengujian ini untuk mengetahui perbandingan performa dengan spesifikasi yang berbeda. Skenario pengujian dapat dilihat pada Tabel 5.6.

Tabel 5.6 Skenario Pengujian (SP-06)

ID	**SP-03**
Nama	Uji Coba Performa Transfer Data Antar 2 Docker *Machine* dari 100 Docker *machine* yan
Tujuan Uji Coba	Perbandingan performa dengan spesifikasi yang berbeda
Kondisi Awal	Terdapat 100 buah Docker *machine* yang terhubung ke dalam 1 switch
Banyak Docker *machine*	100
Metode Pengujian	Uji kecepatan data transfer menggunakan Iperf
Lama Pengujian	15 Detik
Skenario	4. Sambungkan Docker *host* pertama sampai ke seratus ke *switch* 5. Buat *Iperf server* pada Docker *host* pertama 6. Lakukan *Iperf client* pada Docker *host* keseratus
Masukan	-

Keluaran	Kecepatan rata rata dari jaringan
Hasil yang Diharapkan	Kecepatan transfer data akan semakin tinggi
Penggunaan Memori (RAM)	4277 MB
Penggunaan CPU	CPU1 : 0 ~ 99% CPU2 : 0 ~ 73% CPU3 : 5.7 ~ 88% CPU4 : 0 ~ 84%

5.2.7 Skenario Pengujian (SP-07) – Performa MySQL Query didalam Docker *Machine*

Skenario pengujian ini menguji kinerja dari Docker *machine* yang terpasang database MySql didalamnya. Pengujian dilakukan dengan menjalankan fungsi *query* pada database MySql dimana fungsi tersebut memasukkan 1000 inputan di satu satu tabel dimana inputan tersebut memasukkan data pada satu *field* dalam database Pengujian ini dilakukan untuk menguji performa dari query MySql ketika dijalankan dari Docker *machine*. Skenario pengujian dapat dilihat pada Tabel 5.7.

Tabel 5.7 Skenario Pengujian (SP-07)

ID	**SP-07**
Nama	Uji Coba Query MySql didalam Docker
Tujuan Uji Coba	Menguji kinerja query dari MySql
Kondisi Awal	Terdapat Docker *machine* yang menjalankan MySql database
Metode Pengujian	Uji query dengan memasukkan 1000 data kedalam tabel
Lama Pengujian	-
Skenario	1. Nyalakan Docker *Machine,* jalankan MySql dan masuk kedalam MySql CLI 2. Buat fungsi inputan ke MySql

	3. Jalankan fungsi tersebut
Masukan	Fungsi MySql
Keluaran	Kecepatan query dari fungsi inputan
Hasil yang Diharapkan	-

5.2.8 Skenario Pengujian (SP-08) – Performa MySQL Query didalam Docker *Machine* dengan *Sysbench*

Skenario pengujian ini menguji kinerja dari Docker *machine* yang terpasang database MySql didalamnya. Pengujian ini dilakukan untuk menguji performa dari MySql secara keseluruhan dengan menggunakan *sysbench* [13] ketika dijalankan dari Docker *machine*. Pertama *sysbench* melakukan pengisian data dengan menjalankan fungsi dengan parameter-parameter yang telah ditentukan. Kemudian dengan parameter yang sama akan dilakukan pengujian yang diatur oleh *sysbench* sendiri. Skenario pengujian dapat dilihat pada Tabel 5.8

Tabel 5.8 Skenario Pengujian (SP-08)

ID	**SP-08**
Nama	Uji Coba *sysbench* MySql didalam Docker
Tujuan Uji Coba	Menguji kinerja dari MySql
Kondisi Awal	Terdapat Docker *machine* yang menjalankan MySql database
Metode Pengujian	Uji MySql menggunakan *Sysbench*
Lama Pengujian	-
Skenario	1. Nyalakan Docker *Machine,* jalankan MySql dan masuk kedalam MySql CLI 2. Masukkan konfigurasi data *sysbench* kedalam tabel database 3. Jalankan *sysbench*
Masukan	Data testing dari *sysbench*

Keluaran	Keluaran hasil testing dari *sysbench*
Hasil yang Diharapkan	-

5.2.9 Skenario Pengujian (SP-09) – Performa *File IO* Docker *Machine* dengan *Sysbench*

Skenario pengujian ini menguji kinerja *file IO* atau *read* dan *write* dari Docker *machine*. Pengujian ini dilakukan untuk menguji performa *file IO* secara keseluruhan dengan menggunakan *sysbench* ketika dijalankan dari Docker *machine*. Pertama *sysbench* melakukan pembuatan *dummy file,* penulis membuat 8GB. *Dummy file* harus berukuran lebih dari RAM supaya tidak masuk kedalam *cache* dari RAM karena ukurannya yang lebih besar. Kemudian dilakukan pengetesan *read* dan *write* oleh *sysbench* sendiri. Dalam pengetesan ini digunakan Docker *image* Ubuntu:trusty. Skenario pengujian dapat dilihat pada Tabel 5.9.

Tabel 5.9 Skenario Pengujian (SP-09)

ID	**SP-06**
Nama	Uji Coba *sysbench file IO* didalam Docker
Tujuan Uji Coba	Menguji kinerja *file IO* dari Docker
Kondisi Awal	Terdapat Docker *machine* yang berjalan
Metode Pengujian	Uji *file IO* dari Docker menggunakan *Sysbench*
Lama Pengujian	-
Skenario	1. Nyalakan Docker *Machine,* jalankan Ubuntu dan masuk ke bash. 2. Masukkan konfigurasi dari *dummy file* di *sysbench.* 3. Jalankan *sysbench.* 4. Hapus *dummy file.*
Masukan	*Dummy file* dari *sysbench*
Keluaran	Keluaran hasil testing dari *sysbench*

Hasil yang Diharapkan	-

5.2.10 Skenario Pengujian (SP-10) – Stress Test MySQL didalam Docker *Machine* Menggunakan *MySQLSlap*

Skenario pengujian ini menguji stress test dari Docker *machine* yang terpasang database MySql didalamnya. Pengujian dilakukan dengan menjalankan *MySQLSlap*[14] dengan menambahkan jumbah dari *query*. Pengujian ini dilakukan untuk menguji ketahanan dari MySql ketika dijalankan dari Docker *machine*. Skenario pengujian dapat dilihat pada Tabel 5.10.

Tabel 5.10 Skenario Pengujian (SP-10)

ID	**SP-10**
Nama	Stress Test MySql didalam Docker menggunakan *MySQLSlap*
Tujuan Uji Coba	Menguji ketahanan dari MySql
Kondisi Awal	Terdapat Docker *machine* yang menjalankan MySql *database*
Metode Pengujian	Uji *MySQLSlap* dengan menambahkan jumlah *query* setiap *run* nya
Lama Pengujian	-
Skenario	1. Nyalakan Docker *Machine,* jalankan MySql *2.* Buat perintah *MySQLSlap* 3. Jalankan perintah tersebut
Masukan	Fungsi MySql
Keluaran	Kecepatan query dari fungsi inputan
Hasil yang Diharapkan	-

5.3 Hasil Pengujian Performa dan Evaluasi

Subbab ini berisi tentang hasil dari pengujian yang telah dilakukan. Pada subbab ini akan berisi hasil dari masing masing skenario pengujian yang dijelaskan pada subbab sebelumnya.

5.3.1 Hasil Pengujian (SP-01) – Stress Test Jumlah Docker *Machine* (4GB RAM, 1 Core CPU)

Sesuai Skenario SP-01, Hasil yang didapatkan dari uji coba yang dilakukan jumlah banyaknya Docker *machine* yang dapat dijalankan di dalam lingkungan dengan spesifikasi 4GB RAM, 1 Core CPU. Hasil pengujian dapat dilihat pada Tabel 5.11.

Tabel 5.11 Hasil Pengujian (SP-01)

Running ke	**Jumlah Docker *machine***
1	662
2	658
3	594
4	639
5	581
6	609
7	607
8	607
9	655
10	625
Rata Rata	619

5.3.2 Hasil Pengujian (SP-02) – Performa TCP antar 2 Docker *Machine* yang Terhubung di 1 *Switch* (4GB RAM, 1 Core CPU)

Sesuai Skenario SP-02, Hasil yang didapatkan dari uji coba yang dilakukan adalah kecepatan transfer data dari dua Docker

machine yang terhubung ke dalam satu *switch* di dalam sistem *Virtual Data Center* dengan protokol TCP. Hasil pengujian dapat dilihat pada Tabel 5.12.

Tabel 5.12 Hasil Pengujian (SP-02)

Detik ke -	**Transfer (GBytes)**	**Kecepatan (Gbits/sec)**
0 – 1	3.37	28.9
1 – 2	3.38	29.1
2 – 3	3.35	28.8
3 – 4	3.32	28.5
4 – 5	3.19	27.4
5 – 6	3.29	28.3
6 – 7	3.35	28.8
7 – 8	3.38	29.0
8 – 9	3.26	28.0
9 – 10	3.33	28.6
10 – 11	3.36	28.8
11 – 12	3.44	29.6
12 – 13	3.35	28.8
13 – 14	3.37	29.0
14 – 15	3.39	29.1
	Total : 50.2	Rata Rata : 28.7

5.3.3 Hasil Pengujian (SP-03) – Performa TCP antar 2 Docker *Machine* dengan 100 Docker *Machine* Terhubung di 1 Switch (4GB RAM, 1 Core CPU)

Sesuai Skenario SP-03, Hasil yang didapatkan dari uji coba yang dilakukan adalah kecepatan transfer data dari dua Docker *machine* dengan serratus Docker *machine* terhubung ke dalam satu *switch* di dalam sistem *Virtual Data Center* dengan protokol TCP. Terlihat dari hasil dibawah, jumlah Docker *machine* sangat

mempengaruhi hasil dari pengujin ini. Hasil pengujian dapat dilihat pada Tabel 5.13.

Tabel 5.13 Hasil Pengujian (SP-03)

Detik ke -	Transfer (GBytes)	Kecepatan (Gbits/sec)
0 – 1	1.59	13.7
1 – 2	1.58	13.5
2 – 3	1.58	13.6
3 – 4	1.50	12.9
4 – 5	1.44	12.4
5 – 6	1.32	11.3
6 – 7	1.47	12.6
7 – 8	1.46	12.6
8 – 9	1.61	13.9
9 – 10	1.45	12.4
10 – 11	1.48	12.7
11 – 12	1.07	9.19
12 – 13	1.13	9.67
13 – 14	1.56	13.4
14 – 15	1.54	13.2
	Total : 21.8	Rata Rata : 12.1

5.3.4 Hasil Pengujian (SP-04) – Stress Test Jumlah Docker *Machine* (8 GB RAM, 4 Core CPU)

Sesuai Skenario SP-04, Hasil yang didapatkan dari uji coba yang dilakukan jumlah banyaknya Docker *machine* yang dapat dijalankan di dalam lingkungan sesuai dengan spesifikasi 8GB RAM dan 4 Core CPU. Hasil pengujian dapat dilihat pada Tabel 5.14.

Tabel 5.14 Hasil Pengujian (SP-04)

Running ke	**Jumlah Docker *machine***
1	823
2	790
3	793
4	787
5	758
6	792
7	741
8	801
9	735
10	755
Rata Rata	7

5.3.5 Hasil Pengujian (SP-05) – Performa TCP antar 2 Docker *Machine* yang Terhubung di 1 *Switch* (8GB RAM, 4 Core CPU)

Sesuai Skenario SP-05, Hasil yang didapatkan dari uji coba yang dilakukan adalah kecepatan transfer data dari dua Docker *machine* yang terhubung ke dalam satu *switch* di dalam sistem *Virtual Data Center* dengan protokol TCP. Spesifikasi pengujian dengan 8GB RAM dan 4 Core CPU. Hasil pengujian dapat dilihat pada Tabel 5.15.

Tabel 5.15 Hasil Pengujian (SP-05)

Detik ke -	**Transfer (GBytes)**	**Kecepatan (Gbits/sec)**
0 – 1	5.24	45.0
1 – 2	5.29	45.4
2 – 3	4.28	36.7
3 – 4	5.51	47.3

Detik ke -	Transfer (GBytes)	Kecepatan (Gbits/sec)
4 – 5	5.13	44.1
5 – 6	5.35	46.0
6 – 7	5.02	43.1
7 – 8	5.11	43.9
8 – 9	5.10	43.8
9 – 10	5.40	46.3
10 – 11	5.21	44.8
11 – 12	5.31	45.6
12 – 13	5.31	45.6
13 – 14	5.28	45.3
14 – 15	5.44	46.7
	Total : 78.0	Rata Rata : 44.6

5.3.6 Hasil Pengujian (SP-06) – Performa TCP antar 2 Docker *Machine* dengan 100 Docker *Machine* Terhubung di 1 Switch (8GB RAM, 4 Core CPU)

Sesuai Skenario SP-06, Hasil yang didapatkan dari uji coba yang dilakukan adalah kecepatan transfer data dari dua Docker *machine* dengan serratus Docker *machine* terhubung ke dalam satu *switch* di dalam sistem *Virtual Data Center* dengan protokol TCP. Terlihat dari hasil dibawah, jumlah Docker *machine* ternyata tidak terlalu mempengaruhi kecepatan jika sistem tidak dalam kondisi *stress*. Spesifikasi pengujian dengan 8GB RAM dan 4 Core CPU. Hasil pengujian dapat dilihat pada Tabel 5.16.

Tabel 5.16 Hasil Pengujian (SP-06)

Detik ke -	Transfer (GBytes)	Kecepatan (Gbits/sec)
0 – 1	5.18	44.5
1 – 2	4.79	41.2
2 – 3	4.78	41.0

Detik ke -	Transfer (GBytes)	Kecepatan (Gbits/sec)
3 – 4	4.79	41.1
4 – 5	4.97	42.7
5 – 6	4.93	42.3
6 – 7	4.95	42.5
7 – 8	5.00	43.0
8 – 9	5.07	43.6
9 – 10	4.98	42.7
10 – 11	4.74	40.7
11 – 12	4.88	41.9
12 – 13	4.96	42.6
13 – 14	5.07	43.6
14 – 15	4.95	42.5
	Total : 74.1	Rata Rata : 42.4

5.3.7 Hasil Pengujian (SP-07) – Performa MySQL Query didalam Docker *Machine*

Sesuai skenario SP-07, Hasil yang didapatkan dari uji coba yang dilakukan dengan menjalankan fungsi *query* pada database MySql dimana fungsi tersebut memasukkan 1000 inputan di satu satu tabel dimana inputan tersebut memasukkan data pada satu *field* dalam database. Hasil pengujian dapat dilihat pada Tabel 5.17.

Tabel 5.17 Hasil Pengujian (SP-07)

Running ke	Lama waktu query (detik / second)
1	0.96
2	0.96
3	1.45
4	0.83
5	0.83

Running ke	Lama waktu query (detik / second)
6	1.08
7	0.86
8	0.85
9	0.86
10	0.84
Rata Rata	1.123

5.3.8 Hasil Pengujian (SP-08) – Performa MySQL Query didalam Docker *Machine* dengan *Sysbench*

Sesuai skenario SP-08, Hasil yang didapatkan dari uji coba yang dilakukan dengan menguji performa dari MySql secara keseluruhan dengan menggunakan *sysbench*. Pertama *sysbench* melakukan pengisian data dengan menjalankan fungsi dengan parameter-parameter yang telah ditentukan. Kemudian dengan parameter yang sama akan dilakukan pengujian yang kemudian dapat kita lihat hasil dari datanya. Hasil pengujian dapat dilihat pada Tabel 5.18.

Tabel 5.18 Hasil Pengujian (SP-08)

Tes	Nilai
OLTP statistik tes :	
Jumlah query yang dilakukan:	
Read (Jumlah query) *Write* (Jumlah query) *Other* (Jumlah query) Jumlah	3536288 1221609 491162 5249059
Transaksi (Jumlah)	238570 (397.58 per detik)
Deadlocks (Jumlah)	14022 (23.37 per detik)

Tes	Nilai
Permintaan *read/write* (Jumlah)	4757897 (7929.15 per detik)
Operasi yang lain (Jumlah)	491162 (818.53 per detik)
Rangkuman tes :	
Waktu eksekusi (detik)	600.0516 detik
Jumlah dari *event* (Jumlah)	238570
Total waktu dari eksekusi *event* (detik)	23998.5074 detik
Statistik *pre-request* :	
Minimum (millisecond) Rata rata (millisecond) Maksimum (millisecond) Perkiraan. 95 persen dari total (millisecond)	2.20ms 100.59ms 1072.63ms 189.43ms
Thead Fairness :	
Events (Rata-rata/*stddev*)	5964.2500/49.48
Waktu eksekusi (Rata-rata/*stddev*)	599.9627/0.06

5.3.9 Hasil Pengujian (SP-09) – Performa *File IO* Docker *Machine* dengan *Sysbench*

Sesuai Skenario SP-09, Hasil yang didapatkan dari uji coba yang dilakukan dengan menguji performa *file IO* secara keseluruhan dengan menggunakan *sysbench* ketika dijalankan dari Docker *machine*. Pertama *sysbench* melakukan pembuatan *dummy file*. Kemudian akan dilakukan pengujian *file IO* terhadap *dummy file* tersebut yang kemudian dapat kita lihat hasil dari datanya. Hasil pengujian dapat dilihat pada Tabel 5.19.

Tabel 5.19 Hasil Pengujian (SP-09)

Tes	Nilai
Operasi yang dilakukan :	
Read (Jumlah Operasi)	141688

Tes	Nilai
Write (Jumlah Operasi)	94458
Operasi yang lain (Jumlah Operasi)	302208
Total *read* (Gb)	2.162Gb
Total *write* (Gb)	1.4413Gb
Total data ditransfer (Gb)	3.6033Gb
Kecepatan (Mb/detik)	12.299Mb/detik
Kecepatan permintaan dieksekusi (permintaan/detik)	787.15 Permintaan/detik
Rangkuman tes :	
Waktu eksekusi	300.0012 detik
Jumlah dari *event*	236146
Total waktu dari eksekusi *event*	109.3988 detik
Statistik *pre-request* :	
Minimum (millisecond) Rata rata (millisecond) Maksimum (millisecond) Perkiraan. 95 persen dari total (millisecond)	0.00ms 0.46ms 210.76ms 0.15ms
Thead Fairness :	
Events (Rata-rata/*stddev*)	236146.0000/0.00
Waktu eksekusi (Rata-rata/*stddev*)	109.3988/0.00

5.3.10 Hasil Pengujian (SP-10) – Stress Test MySQL didalam Docker *Machine* Menggunakan *MySQLSlap*

Sesuai Skenario SP-10, Hasil yang didapatkan dari uji coba yang dilakukan dengan menjalankan *MySQLSlap* di dalam MySql dimana jumlah *query* yang berjalan pada *MySQLSlap* akan ditambahkan secara bertahap . Hasil pengujian dapat dilihat pada Tabel 5.20.

Tabel 5.20 Hasil Pengujian (SP-010)

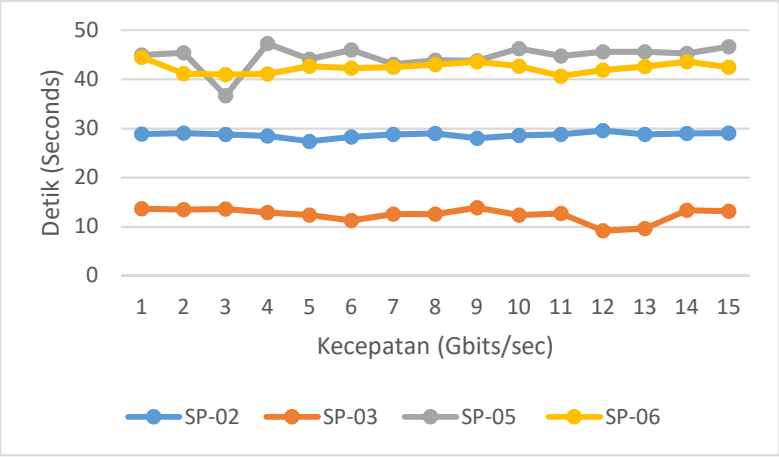
Jumlah Query	Waktu Minimal (detik / second)	Waktu Maksimal (detik / second)	Waktu Rata-Rata (detik / second)
1000	0.46	0.67	0.57
2000	1.29	2.14	1.63
3000	2.54	3.09	2.79
4000	3.98	4.59	4.31
5000	5.63	6.17	5.83
6000	7.74	8.93	8.21
7000	10.07	12.61	10.90
8000	13.24	14.75	14.13
9000	15.76	17.97	16.83
10000	19.15	21.29	20.13

5.3.11 Evaluasi

Dari hasil data pengujian diatas dapat kita lihat bahwa dengan spesifikasi lingkungan sistem yang tertera di Bab 5.1. Lingkungan pengujian dapat menjalankan rata-rata 161 Docker *machine* secara langsung dengan kondisi Docker *machine* tidak tersambung ke *switch* dan sistem *Virtual Data Center.* Semakin tinggi spesifikasi dari lingkunagan sistem, semakin banyak juga Docker machine yang dapat berjalan. Lihat Tabel 5.11 dan Tabel 5.14

Kemudian untuk kecepatan data transfer antar Docker *host machine* yang terhubung ke dalam satu *switch* sangat dipengaruhi oleh jumlah Docker *host machine* yang terhubung kedalam *switch* tersebut. Dari pengujian kecepatan data transfer tersebut juga kita dapat melihat jumlah RAM dan CPU yang termakan untuk jumlah Docker *host machine* yang berjalan. Untuk dua Docker dan satu *switch* menghabiskan RAM sebesar 260MB

dengan konsumsi CPU kurang lebih ~2%. Sedangkan untuk 100 Docker dan satu *switch* menghabiskan hamper seluruh RAM dan CPU dengan penggunaan RAM sebesar 3800MB dari 4GB RAM yang tersedia dan 100% penggunaan CPU, yang dimana merupakan batas dari lingkungan pengujian ditandakan dengan terjadinya *lag* didalam lingkungan pengujian. Dari hasil perbandingan yang dilakukan pada lingkungan pengujian dapat diambil kesimpulan bahwa kecepatan transfer sangat dipengaruhi oleh spesifikasi dari sistem pengujian. Semakin *stress* sistem semakin lambat pula kecepatan transfer yang diperoleh. Lihat Tabel 5.12 dan Tabel 5.13. Dan juga Tabel 5.15 dan Tabel 5.16 untuk perbandingan dengan spesifikasi yang berbeda. Gambar 5.4 merupkan grafik untuk perbandingan dari pengujian kecepatan SP-02, SP-03, SP-05, dan SP-06.

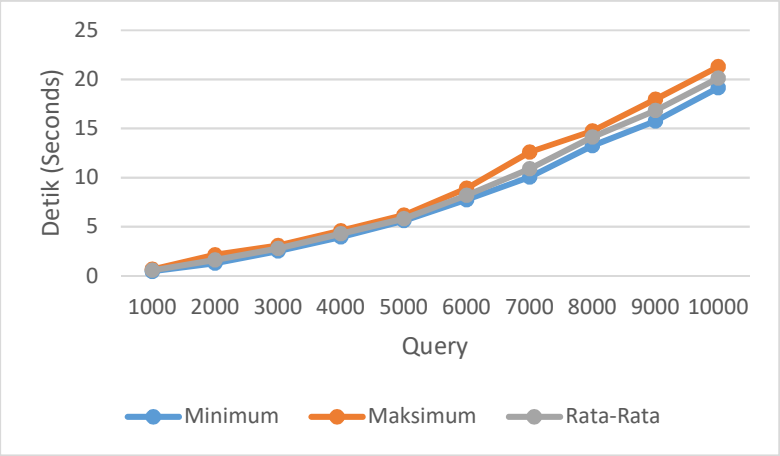


Gambar 5.4 Grafik Perbandingan Kecepatan Hasil dari SP-02, SP-03, SP-05, dan SP-06

Untuk uji performa MySQL Docker *machine* berhasil melakukan 1000 *query* dalam waktu rata-rata 1.123 detik. Hasil dari uji performa MySQL *sysbench* dengan batas waktu 600 detik berhasil melakukan 5249059 query. Untuk uji performa *file IO*

sysbench dengan ukuran file 8GB dan waktu 300 detik. Lihat Tabel 5.17, Tabel 5.18, dan Tabel 5.19.

Untuk hasil dari uji stress test dari MySQL menguunakan MySQLSlap dapat dilihat pada Tabel 5.20 dan juga garfik di Gambar 5.5.



Gambar 5.5 Grafik Hasil Stress Test MySQL menggunakan *MySQlSlap*

5.4 Skenario Pengujian Fungsionalitas

Diperlukan beberapa skenario untuk menguji sistem yang telah dibuat. Skenario pengujian performa diperlukan untuk menguji bekerja maupun tidaknya sistem yang telah dibuat dalam menghadapi kemungkinan yang terjadi dalam penggunaan nyata dari sistem.

5.4.1 Skenario Pengujian (SP-11) – Menambahkan *Host* Docker *Container*

Uji coba ini dijalankan dengan tujuan untuk menguji fungsionalitas dari website untuk menambahkan *host* baru.

Pengujian fungsionalitas ini diawali dengan *login* pada website terlebih dahulu. Tabel 5.22 adalah tabel mengenai skenario uji coba untuk menambahkan *host* Docker.

Tabel 5.21 Hasil Pengujian (SP-11)

ID	**SP-11**
Nama	Uji Coba Menambahkan *host* Docker.
Tujuan Uji Coba	Menguji fungsionalitas website untuk menambahkan *host* Docker di dalam sistem *Virtual Data Center*.
Kondisi Awal	Membuka halaman website
Skenario	1. Membuka halaman website 2. *Login* sebagai Pengguna 3. Masuk ke halaman *Add Docker* 4. Mengisi informasi yang diperlukan 5. Tekan tombol *Add Docker*
Masukan	Informasi yang diperlukan.
Keluaran	*Host* Docker baru
Hasil yang Diharapkan	Menambah *host* Docker dan menampilkan info Docker tersebut di tabel *list host*.

5.4.2 Skenario Pengujian (SP-12) – Mengubah Status *Host* Docker *Container*

Uji coba ini dijalankan dengan tujuan untuk menguji fungsionalitas dari website untuk mengubah status dari *host* yang telah ada. Pengujian fungsionalitas ini diawali dengan *login* pada website terlebih dahulu. Tabel 5.22 adalah tabel mengenai skenario uji coba untuk menambahkan *host* Docker.

Tabel 5.22 Hasil Pengujian (SP-12)

ID	**SP-12**
Nama	Uji Coba Mengubah Status *Host* Docker.
Tujuan Uji Coba	Menguji fungsionalitas website untuk mengubah *host* Docker di dalam sistem *Virtual Data Center*.
Kondisi Awal	Membuka halaman website

Skenario	1. Membuka halaman website 2. *Login* sebagai Pengguna 3. Masuk ke halaman utama 4. Tekan tombol *UP*/*DOWN*
Masukan	-
Keluaran	Berubahnya status *host* Docker
Hasil yang Diharapkan	Berubahnya status *host* Docker dan menampilkan info Docker tersebut di tabel *list host*.

5.4.3 Skenario Pengujian (SP-13) – Menghapus *Host* Docker *Container*

Uji coba ini dijalankan dengan tujuan untuk menguji fungsionalitas dari website untuk menghapus *host* yang telah ada. Pengujian fungsionalitas ini diawali dengan *login* pada website terlebih dahulu. Tabel 5.23 adalah tabel mengenai skenario uji coba untuk menambahkan *host* Docker.

Tabel 5.23 Hasil Pengujian (SP-13)

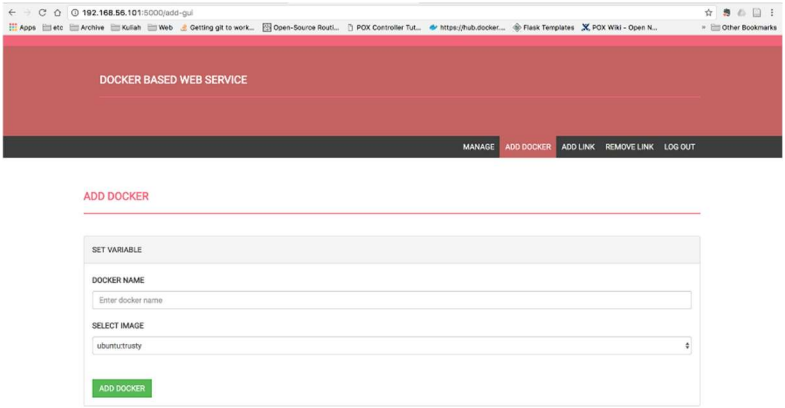
ID	**SP-13**
Nama	Uji Coba Menghapus *Host* Docker.
Tujuan Uji Coba	Menguji fungsionalitas website untuk menghapus *host* Docker di dalam sistem *Virtual Data Center*.
Kondisi Awal	Membuka halaman website
Skenario	1. Membuka halaman website 2. *Login* sebagai Pengguna 3. Masuk ke halaman utama 4. Tekan tombol *Delete*
Masukan	-
Keluaran	Terhapusnya *host* Docker
Hasil yang Diharapkan	Terhapusnya *host* Docker dan memperbaharui info tabel *list host*.

5.5 Hasil Pengujian Fungsionalitas

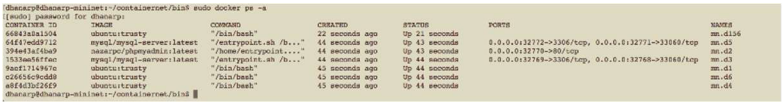
Subbab ini berisi tentang hasil dari pengujian yang telah dilakukan. Pada subbab ini akan berisi hasil dari masing masing skenario pengujian yang dijelaskan pada subbab sebelumnya.

5.5.1 Hasil Pengujian (SP-11) – Menambahkan *Host* Docker *Container*

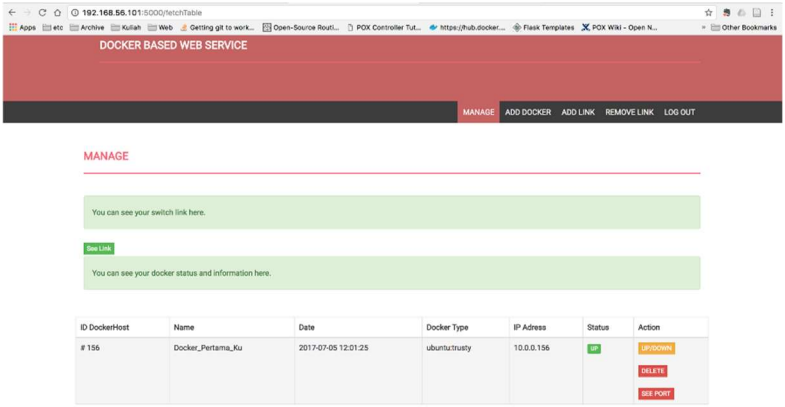
Menurut skenario pada SP-11 , Pengguna pada awalnya akan *login* ke dalam website dengan menggunakan *username* dan *password* yang telah dimiliki. Setelah *login*, pengguna dapat langsung menuju halaman *Add Docker*. Pada uji coba yang dihasilkan dapat dilihat pada Gambar 5.6, Gambar 5.7, danGambar 5.8. Hal ini menunjukkan bahwa fungsionalitas dari SP-11 berjalan dengan baik.



Gambar 5.6 Halaman *Add Docker*



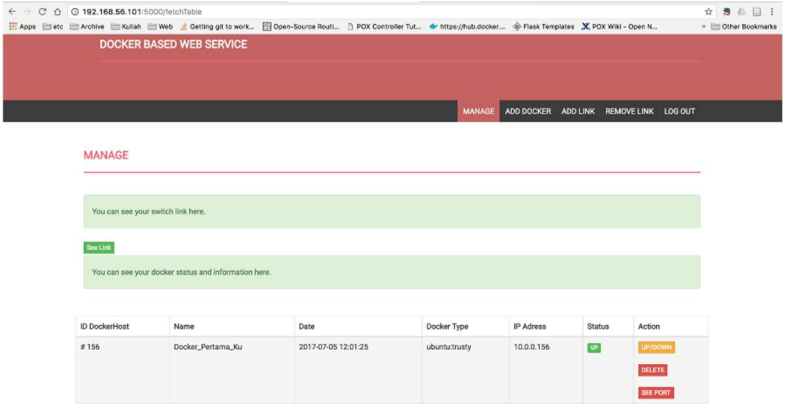
Gambar 5.7 List Docker



Gambar 5.8 Tabel *List Host* Docker

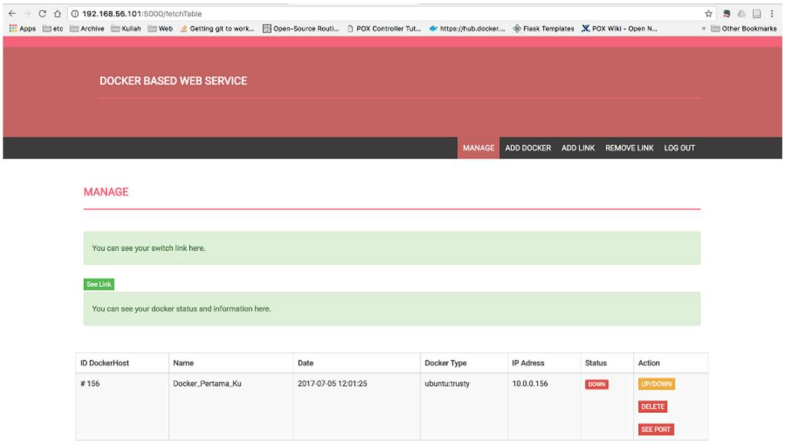
5.5.2 Hasil Pengujian (SP-12) – Mengubah *Host* Docker *Container*

Menurut skenario pada SP-11 , Pengguna pada awalnya akan *login* ke dalam website dengan menggunakan *username* dan *password* yang telah dimiliki. Setelah *login*, pengguna disuguhkan halaman utama yang berisi tabel informasi dari *host* Docker miliknya. Untuk dapat mengubah status dari *host* Docker pengguna dapat memencet tombol *UP* / *DOWN* di tabel info tersebut. Pada uji coba yang dihasilkan dapat dilihat pada Gambar 5.9, Gambar 5.10, dan Gambar 5.11. Hal ini menunjukkan bahwa fungsionalitas dari SP-12 berjalan dengan baik.



Gambar 5.9 Tabel *List Host* Docker

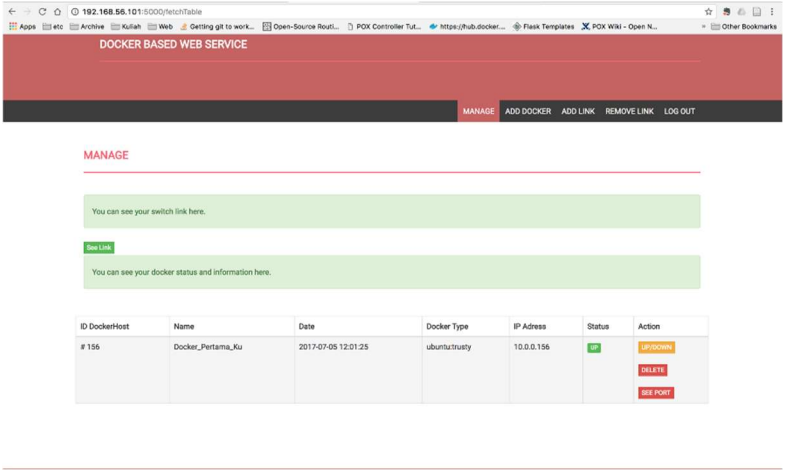
Gambar 5.10 Berubahnya Status Docker Menjadi *Exited*



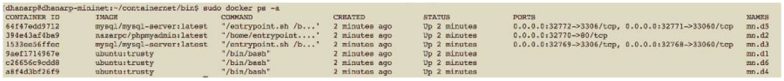
Gambar 5.11 Berubahnya Status *Host* Docker Menjadi *DOWN*

5.5.3 Hasil Pengujian (SP-13) – Mengubah *Host* Docker *Container*

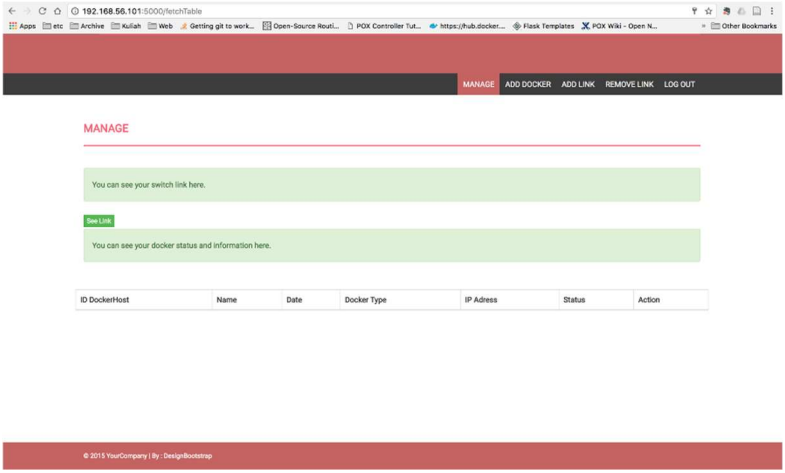
Menurut skenario pada SP-11 , Pengguna pada awalnya akan *login* ke dalam website dengan menggunakan *username* dan *password* yang telah dimiliki. Setelah *login*, pengguna disuguhkan halaman utama yang berisi tabel informasi dari *host* Docker miliknya. Untuk dapat menghapus *host* Docker pengguna dapat memencet tombol *Delete* di tabel info tersebut. Pada uji coba yang dihasilkan dapat dilihat pada Gambar 5.12, Gambar 5.13, dan Gambar 5.14. Hal ini menunjukkan bahwa fungsionalitas dari SP-13 berjalan dengan baik.



Gambar 5.12 Tabel *List Host* Docker



Gambar 5.13 Docker Telah Dihapus



Gambar 5.14 *Host* Docker Telah Dihapus

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan perangkat lunak nantinya.

5.6 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba Performa dari Docker di sistem *Virtual Data Center* dengan uji *stress test, iperf dan sysbench* adalah sebagai berikut:

1. Docker dapat diimplementasikan dan dikelola dengan menggunakan platform Containernet yang dimana didasari dari platform Mininet.
2. Performa dari sistem *Virtual Data Center* dipengaruhi oleh jumlah Docker *machine* yang menyala dan jumlah koneksi ke *switch.*
3. Untuk performa MySql, Docker di dalam sistem *Virtual Data Center* berhasil melakukan kurang lebih 8748 *query* per detik.
4. Untuk performa *file IO*, Docker di dalam sistem *Virtual Data Center* berhasil mencetak kecepatan *read/write* sebesar 12.299Mb/detik.

5.7 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah membuat Docker *image* khusus yang ringan dari DockerFile untuk *image* yang tidak memiliki *package net-tools* dan *iproute* seperti phpmyadmin dan MySql yang telah disediakan di library Docker.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [Daring]. Tersedia pada: https://www.opennetworking.org/sdn-resources/sdn-definition. [Diakses: 27-Apr-2017].

[2] "OpenFlow » What is OpenFlow?" .

[3] "What is Network Virtualization? - Definition," *SDxCentral*, 25-Agu-2013. [Daring]. Tersedia pada: https://www.sdxcentral.com/sdn/network-virtualization/definitions/whats-network-virtualization/. [Diakses: 02-Mei-2017].

[4] "Mininet Overview - Mininet." [Daring]. Tersedia pada: http://mininet.org/overview/. [Diakses: 02-Mei-2017].

[5] "What is a Container," *Docker*, 29-Jan-2017. [Daring]. Tersedia pada: https://www.docker.com/what-container. [Diakses: 02-Mei-2017].

[6] "mpeuster/containernet - Docker Hub." [Daring]. Tersedia pada: https://hub.docker.com/r/mpeuster/containernet/. [Diakses: 02-Mei-2017].

[7] "Welcome to Python.org," *Python.org*. [Daring]. Tersedia pada: https://www.python.org/doc/essays/blurb/. [Diakses: 02-Mei-2017].

[8] "noxrepo/pox," *GitHub*. [Daring]. Tersedia pada: https://github.com/noxrepo/pox. [Diakses: 02-Mei-2017].

[9] "Welcome | Flask (A Python Microframework)." [Daring]. Tersedia pada: http://flask.pocoo.org/. [Diakses: 03-Mei-2017].

[10] "What is HTML5? - Definition from Techopedia," *Techopedia.com*. [Daring]. Tersedia pada: https://www.techopedia.com/definition/1891/html5. [Diakses: 03-Mei-2017].

[11] "Bootstrap Get Started." [Daring]. Tersedia pada: https://www.w3schools.com/bootstrap/bootstrap_get_started.asp. [Diakses: 03-Mei-2017].

[12] “iPerf - The TCP, UDP and SCTP network bandwidth measurement tool.” [Daring]. Tersedia pada: https://iperf.fr/. [Diakses: 28-Mei-2017].
[13] “How To Benchmark Your System (CPU, File IO, MySQL) With sysbench,” *Howtoforge*. [Daring]. Tersedia pada: https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench. [Diakses: 28-Mei-2017].
[14] “MySQL :: MySQL 5.7 Reference Manual :: 4.5.9 mysqlslap — Load Emulation Client.” [Daring]. Tersedia pada: https://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html. [Diakses: 04-Jul-2017].

LAMPIRAN

8.1 HTML WEB UI

Kode Sumber 8.1 login-gui.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
    <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}" rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}"  rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    <header>
    </header>
    <!-- HEADER END-->
    <div class="navbar navbar-inverse set-radius-zero">
        <div class="container">
```

```
            <div class="navbar-header">
                <button type="button" class="navbar-
toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </div>

            <div class="left-div">
                <div class="user-settings-wrapper">
                    <ul class="nav">

                        <h4 class="page-head-line"
style="color:white">Docker Based Web Service</h4>

                    </ul>
                </div>
            </div>
        </div>
    </div>
    <!-- LOGO HEADER END-->

    <!-- MENU SECTION END-->
    <div class="content-wrapper">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <h4 class="page-head-line">Please
Login or Register To Enter </h4>
                    {% if error %}
                        <p
class=error><strong>Status:</strong> {{ error }}
                    {% endif %}
                </div>

            </div>
            <div class="row">
                <form action="/signup" method="POST"
class="col-md-6">
                  <h4> Register </h4>
                   <br />
                   <label>Enter Email ID : </label>
                       <input type="text" name="username"
class="form-control" />
                       <label>Enter Password :  </label>
                       <input type="password"
name="password" class="form-control" />
                       <hr />
```

```
                        <button  class="btn btn-
warning"><span class="glyphicon glyphicon-user"></span>
 Register </button> 
                    <br /><br />
                     <div class="alert alert-info">
                        You can directly enter if you
already have an acount or register if you don't. Enjoy our
services
                        <br />

                    </div>
                </form>
                <form class="col-md-6" action="/login-gui"
method="POST">
                    <h4> Or Login</h4>
                    <br />
                     <label>Enter Email ID : </label>
                        <input type="text" class="form-
control" name="username"/>
                        <label>Enter Password :  </label>
                        <input type="password"
class="form-control" name="password"/>
                        <hr />
                        <button type="Submit"
value="login" class="btn btn-info"><span class="glyphicon
glyphicon-user"></span>  Log Me In </button> 
                </form>

            </div>
        </div>
    </div>
    <!-- CONTENT-WRAPPER SECTION END-->
    <footer>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    &copy; 2015 YourCompany | By : <a
href="http://www.designbootstrap.com/"
target="_blank">DesignBootstrap</a>
                </div>

            </div>
        </div>
    </footer>
    <!-- FOOTER SECTION END-->
    <!-- JAVASCRIPT AT THE BOTTOM TO REDUCE THE LOADING
TIME  -->
    <!-- CORE JQUERY SCRIPTS -->
```

```
    <script src="{{ url_for('static', filename='js/jquery-
1.11.1.js') }}" type="text/javascript"></script>
    <!-- BOOTSTRAP SCRIPTS  -->
    <script src="{{ url_for('static',
filename='js/bootstrap.js') }}"
type="text/javascript"></script>
</body>
</html>
```

Kode Sumber 8.2 index-gui.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
    <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    <header>
```

```
    </header>
    <!-- HEADER END-->
    <div class="navbar navbar-inverse set-radius-zero">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-
toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </div>

            <div class="left-div">
                <div class="user-settings-wrapper">
                    <ul class="nav">

                        <h4 class="page-head-line"
style="color:white">Docker Based Web Service</h4>

                    </ul>
                </div>
            </div>
        </div>
    </div>
    <!-- LOGO HEADER END-->
    <section class="menu-section">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <div class="navbar-collapse collapse
">
                        <ul id="menu-top" class="nav
navbar-nav navbar-right">
                            <li><a class="menu-top-active"
href="i/ndex-gui">MANAGE</a></li>
                            <li><a href="/add-gui">ADD
DOCKER</a></li>
                            <li><a
href="/formtambahlink">ADD LINK</a></li>
                            <li><a
href="/formlinkupdown">REMOVE LINK</a></li>
                            <li><a href="/logout">LOG
OUT</a></li>


                        </ul>
                    </div>
                </div>
```

```
            </div>
        </div>
    </section>
    <!-- MENU SECTION END-->
    <div class="content-wrapper">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <h4 class="page-head-line">MANAGE</h4>

                </div>

            </div>
            <div class="row">
                <div class="col-md-12">
                    <div class="alert alert-success">
                        You can see your switch link here.
                    </div>
                        <a class="btn btn-xs btn-success"
name="seeLink" href="/seelinkofswitch" target="_blank">See
Link</a>
                    <br>
                    <div class="alert alert-success">
                        You can see your docker status and
information here.
                    </div>
                </div>

            </div>
            <br>
            <div class="row">
                 <div class="table-responsive">
                                <table class="table table-
striped table-bordered table-hover">
                                    <thead>
                                        <tr>
                                            <th>ID
DockerHost</th>
                                            <th>Name</th>
                                            <th>Date</th>
                                            <th>Docker
Type</th>
                                            <th>IP
Adress</th>

<th>Status</th>

<th>Action</th>
```

```
                                        </tr>
                                    </thead>
                                    <tbody>
                                        {% for row in rows
%}
                                        <tr>
                                            <td># {{
row[0] }}</td>
                                            <td> {{ row[2]
}}</td>
                                            <td> {{ row[3]
}}</td>
                                            <td> {{ row[5]
}}</td>
                                            <td> {{ row[6]
}}</td>
                                            {% if
row[4]==1 %}
                                                <td>
                                                <label
class="label label-success">UP</label></td>
                                            {% else %}
                                                <td>
                                                <label
class="label label-danger">DOWN</label>
                                                </td>
                                            {% endif %}
                                            <td>
                                                <form
action="/upOrDown" method="POST">

<button class="btn btn-xs btn-warning" name="dockerID"
value="{{row[0]}}">UP/DOWN</button>
                                                </form>
                                                <br>
                                                <form
action="/deleteDocker" method="POST">

<button class="btn btn-xs btn-danger" name="dockerID"
value="{{row[0]}}">DELETE</button>
                                                </form>
                                                <br>
                                                <form
action="/seePort" method="POST" target="_blank">

<button class="btn btn-xs btn-danger" name="dockerID"
value="{{row[0]}}">SEE PORT</button>
                                                </form>
                                            </td>
```

```
                                        </tr>
                                        {% endfor %}
                                        <!--</tr>-->
                                    </tbody>
                                </table>

                            </div>
                        </div>

                        </div>
                    </div>
                </div>
                <!-- CONTENT-WRAPPER SECTION END-->
                <footer>
                    <div class="container">
                        <div class="row">
                            <div class="col-md-12">
                                &copy; 2015 YourCompany | By : <a
href="http://www.designbootstrap.com/"
target="_blank">DesignBootstrap</a>
                            </div>

                        </div>
                    </div>
                </footer>
                <!-- FOOTER SECTION END-->
                <!-- JAVASCRIPT AT THE BOTTOM TO REDUCE THE LOADING
TIME  -->
                <!-- CORE JQUERY SCRIPTS -->
                <script src="{{ url_for('static', filename='js/jquery-
1.11.1.js') }}" type="text/javascript"></script>
                <!-- BOOTSTRAP SCRIPTS  -->
                <script src="{{ url_for('static',
filename='js/bootstrap.js') }}"
type="text/javascript"></script>
</body>
</html>
```

Kode Sumber 8.3 add-gui.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
```

```
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
   <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    <header>
    </header>
    <!-- HEADER END-->
    <div class="navbar navbar-inverse set-radius-zero">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-
toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </div>

            <div class="left-div">
                <div class="user-settings-wrapper">
                    <ul class="nav">
```

```
                        <h4 class="page-head-line"
style="color:white">Docker Based Web Service</h4>

                    </ul>
                </div>
            </div>
        </div>
    </div>
    <!-- LOGO HEADER END-->
    <section class="menu-section">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <div class="navbar-collapse collapse
">
                        <ul id="menu-top" class="nav
navbar-nav navbar-right">
                            <li><a href="/index-
gui">MANAGE</a></li>
                            <li><a class="menu-top-active"
href="/add-gui">ADD DOCKER</a></li>
                            <li><a
href="/formtambahlink">ADD LINK</a></li>
                            <li><a
href="/formlinkupdown">REMOVE LINK</a></li>
                            <li><a href="/logout">LOG
OUT</a></li>

                        </ul>
                    </div>
                </div>

            </div>
        </div>
    </section>
    <!-- MENU SECTION END-->
    <div class="content-wrapper">
        <div class="container">
              <div class="row">
                    <div class="col-md-12">
                        <h1 class="page-head-line">ADD
DOCKER </h1>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-12">
                        <div class="panel panel-default">
                        <div class="panel-heading">
                           SET VARIABLE
```

```
                        </div>
                        <div class="panel-body">
                       <form action="/addDocker"
method="POST">
  <div class="form-group">
    <label for="exampleInputEmail1">DOCKER NAME</label>
    <input type="text" class="form-control"
name="dockerName" placeholder="Enter docker name" />
  </div>
  <div class="form-group">
                                              <label>SELECT
IMAGE</label>
                                              <select
class="form-control" name="dockerImage">
                                                  <option
value="ubuntu:trusty">ubuntu:trusty</option>
                                                  <option
value="mysql/mysql-server:latest">mysql/mysql-
server:latest</option>
                                                  <option
value="nazarpc/phpmyadmin:latest">nazarpc/phpmyadmin:lates
t</option>
                                              </select>
                                          </div>

<br>
<button type="Submit" class="btn btn-success"
value="addDocker">ADD DOCKER</button>

</form>
                            </div>
                            </div>
                    </div>

                </div>
        </div>
    </div>
    <!-- CONTENT-WRAPPER SECTION END-->
    <footer>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    &copy; 2015 YourCompany | By : <a
href="http://www.designbootstrap.com/"
target="_blank">DesignBootstrap</a>
                </div>

            </div>
        </div>
```

```
    </footer>
    <!-- FOOTER SECTION END-->
    <!-- JAVASCRIPT AT THE BOTTOM TO REDUCE THE LOADING
TIME  -->
    <!-- CORE JQUERY SCRIPTS -->
    <script src="{{ url_for('static', filename='js/jquery-
1.11.1.js') }}" type="text/javascript"></script>
    <!-- BOOTSTRAP SCRIPTS  -->
    <script src="{{ url_for('static',
filename='js/bootstrap.js') }}"
type="text/javascript"></script>
</body>
</html>
```

Kode Sumber 8.4 formlinkupdown.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
   <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
```

```
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    <header>
    </header>
    <!-- HEADER END-->
    <div class="navbar navbar-inverse set-radius-zero">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-
toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </div>

            <div class="left-div">
                <div class="user-settings-wrapper">
                    <ul class="nav">

                        <h4 class="page-head-line"
style="color:white">Docker Based Web Service</h4>

                    </ul>
                </div>
            </div>
        </div>
    </div>
    <!-- LOGO HEADER END-->
    <section class="menu-section">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <div class="navbar-collapse collapse
">
                        <ul id="menu-top" class="nav
navbar-nav navbar-right">
                            <li><a href="/index-
gui">MANAGE</a></li>
                            <li><a href="/add-gui">ADD
DOCKER</a></li>
                            <li><a
href="/formtambahlink">ADD LINK</a></li>
                            <li><a class="menu-top-active"
href="/formlinkupdown">REMOVE LINK</a></li>
```

```
                            <li><a href="/logout">LOG
OUT</a></li>

                        </ul>
                    </div>
                </div>

            </div>
        </div>
    </section>
    <!-- MENU SECTION END-->
    <div class="content-wrapper">
        <div class="container">
             <div class="row">
                  <div class="col-md-12">
                      <h1 class="page-head-line">DELETE
LINK </h1>
                  </div>
                </div>
                <div class="row">
                    <div class="col-md-12">
                        <div class="panel panel-default">
                        <div class="panel-heading">
                           SET VARIABLE
                        </div>
                        <div class="panel-body">
                       <form action="/linkupdown"
method="POST">
  <div class="form-group">
    <label for="exampleInputEmail1">Enter Destination
Username</label>
    <input type="text" value="Username of Destination
Switch" class="form-control" name="dest"
placeholder="Enter Destination Username">
  </div>

<br>
<button  class="btn btn-warning">  Delete
Link</button>

</form>
                            </div>
                            </div>
                    </div>

                </div>
        </div>
    </div>
    <!-- CONTENT-WRAPPER SECTION END-->
```

```
    <footer>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    &copy; 2015 YourCompany | By : <a
href="http://www.designbootstrap.com/"
target="_blank">DesignBootstrap</a>
                </div>

            </div>
        </div>
    </footer>
    <!-- FOOTER SECTION END-->
    <!-- JAVASCRIPT AT THE BOTTOM TO REDUCE THE LOADING
TIME  -->
    <!-- CORE JQUERY SCRIPTS -->
    <script src="{{ url_for('static', filename='js/jquery-
1.11.1.js') }}" type="text/javascript"></script>
    <!-- BOOTSTRAP SCRIPTS  -->
    <script src="{{ url_for('static',
filename='js/bootstrap.js') }}"
type="text/javascript"></script>
</body>
</html>
```

Kode Sumber 8.5 formtambahlink.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
   <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
```

```
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    <header>
    </header>
    <!-- HEADER END-->
    <div class="navbar navbar-inverse set-radius-zero">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-
toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
            </div>

            <div class="left-div">
                <div class="user-settings-wrapper">
                    <ul class="nav">

                        <h4 class="page-head-line"
style="color:white">Docker Based Web Service</h4>

                    </ul>
                </div>
            </div>
        </div>
    </div>
    <!-- LOGO HEADER END-->
    <section class="menu-section">
        <div class="container">
            <div class="row">
                <div class="col-md-12">
```

```
                    <div class="navbar-collapse collapse
">
                        <ul id="menu-top" class="nav
navbar-nav navbar-right">
                            <li><a href="/index-
gui">MANAGE</a></li>
                            <li><a href="/add-gui">ADD
DOCKER</a></li>
                            <li><a class="menu-top-active"
href="/formtambahlink">ADD LINK</a></li>
                            <li><a
href="/formlinkupdown">REMOVE LINK</a></li>
                            <li><a href="/logout">LOG
OUT</a></li>

                        </ul>
                    </div>
                </div>

            </div>
        </div>
    </section>
    <!-- MENU SECTION END-->
    <div class="content-wrapper">
        <div class="container">
              <div class="row">
                    <div class="col-md-12">
                        <h1 class="page-head-line">ADD
LINK </h1>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-12">
                        <div class="panel panel-default">
                        <div class="panel-heading">
                           SET VARIABLE
                        </div>
                        <div class="panel-body">
                       <form action="/tambahlink"
method="POST">
  <div class="form-group">
    <label for="exampleInputEmail1">Enter Destination
Username</label>
    <input type="text" value="Username of Destination
Switch" class="form-control" name="dest"
placeholder="Enter Destination Username">
  </div>

<br>
```

```
<button  class="btn btn-success">  ADD LINK</button>

</form>
                            </div>
                            </div>
                    </div>

                </div>
        </div>
    </div>
    <!-- CONTENT-WRAPPER SECTION END-->
    <footer>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    &copy; 2015 YourCompany | By : <a
href="http://www.designbootstrap.com/"
target="_blank">DesignBootstrap</a>
                </div>

            </div>
        </div>
    </footer>
    <!-- FOOTER SECTION END-->
    <!-- JAVASCRIPT AT THE BOTTOM TO REDUCE THE LOADING
TIME  -->
    <!-- CORE JQUERY SCRIPTS -->
    <script src="{{ url_for('static', filename='js/jquery-
1.11.1.js') }}" type="text/javascript"></script>
    <!-- BOOTSTRAP SCRIPTS  -->
    <script src="{{ url_for('static',
filename='js/bootstrap.js') }}"
type="text/javascript"></script>
</body>
</html>
```

Kode Sumber 8.6 seeLink.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
```

```
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
    <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
<table class="table table-striped table-bordered table-
hover">
    <thead>
        <tr>
            <th>Source</th>
            <th>Destination</th>
        </tr>
    </thead>
    <tbody>
        {% for hasil in hasil %}
        <tr>
            <td> {{ hasil[1] }}</td>
            <td> {{ hasil[3] }}</td>
        </tr>
        {% endfor %}
        <!--</tr>-->
    </tbody>
</table>
</body>
</html>
```

Kode Sumber 8.7 seePort.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1, maximum-scale=1" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <!--[if IE]>
        <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1">
        <![endif]-->
    <title>Docker Based Mininet with SDN
Controller</title>
    <!-- BOOTSTRAP CORE STYLE  -->
    <link href="{{ url_for('static',
filename='css/bootstrap.css') }}" rel="stylesheet" />
    <!-- FONT AWESOME ICONS  -->
    <link href="{{ url_for('static', filename='css/font-
awesome.css') }}"rel="stylesheet" />
    <!-- CUSTOM STYLE  -->
    <link href="{{ url_for('static',
filename='css/style.css') }}" rel="stylesheet" />
     <!-- HTML5 Shiv and Respond.js for IE8 support of
HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the
page via file:// -->
    <!--[if lt IE 9]>
        <script
src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv
.js"></script>
        <script
src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
min.js"></script>
    <![endif]-->
</head>
<body>
    {{ output }}
</body>
</html>
```

8.2 Web API (Flask, Celery, Redis)

Kode Sumber 8.8 tes1.py (Flask API dan Celery)

```
from flask import Flask, session, redirect, url for,
escape, request, render_template, jsonify, Response
from hashlib import md5
from base64 import b64encode
import MySQLdb
from subprocess import call, Popen, PIPE
from celery import Celery
import fikri
import lordy


app = Flask(__name__)
db = MySQLdb.connect(host="localhost", user="root",
passwd="matapancing", db="tugasakhir")
cur = db.cursor()
p="a"
# Celery configuration
app.config['CELERY BROKER URL'] = 'redis://0.0.0.0:6379/0'
app.config['CELERY_RESULT_BACKEND'] =
'redis://0.0.0.0:6379/0'

# Initialize Celery
celery = Celery(app.name,
broker=app.config['CELERY BROKER URL'])
celery.conf.update(app.config)


'''
@app.route('/')
def users():
    cur.execute(SELECT * FROM user)
    rv = cur.fetchall()
    return str(rv)
'''

@app.route('/')
def index():
    error = None
    print index
    if 'username' in session:
        return redirect(url for('indexpage'))
    return render_template('login-gui.html', error=error)

@app.route('/index-gui')
def indexpage():
```

```
    error = None
    print index
    if 'username' in session:
        return redirect(url_for('fetchTable'))
    return render_template('login-gui.html', error=error)

@app.route('/logout')
def logout():
    #this is a comments
    error='Berhasil Log Out!'
    session.pop('username', None) #kalo semua pake clear
    return render_template('login-gui.html', error=error)

@app.route('/fetchTable')
def fetchTable():
    rows = None
    error = None
    hasil = None
    print index
    username_form  = session['username']
    #cur.execute("SELECT COUNT(1) FROM user WHERE username
= %s;", [username_form])
    cur.execute("SELECT iduser FROM user WHERE username =
%s;", [username_form])
    hasil=cur.fetchall();
    for hasil1 in hasil:
        idUser_form=str(hasil1[0])

    cur.execute("SELECT * from dockerhost where iduser=
%s;", [idUser_form])
    rows = cur.fetchall()
    #db.commit

    return render_template('index-gui.html', rows=rows,
error=error)

@app.route('/add-gui')
def addgui():
    error = None
    print index
    if 'username' in session:
        return render_template('add-gui.html')
    return render_template('login-gui.html', error=error)

@app.route('/tesnet')
def tesNet():
    return tesNets.delay()

@app.route('/addswitch')
```

```
def lordyAddSwitches():
    source=str(session['username'])
    cur.execute("SELECT iduser FROM user WHERE username =
%s;", [source])
    hasil=cur.fetchall();
    for hasil1 in hasil:
            source_id=str(hasil1[0])
    source_id="s"+str(source_id)
    addSwitchs.delay(source_id)
    return str(source_id)

@app.route('/tesbash')
def tesbash():
    #return str(check output(["mkdir", "duhdek"]),
shell=True)
    #return str(call(["mkdir",  "duhdek"]))
    output = Popen(['ls', '-a'], stdout=PIPE)
    return output.stdout.read()

#untuk restart jaringan
@app.route('/tescelery')
def tescelery():
    myNetworks.delay()
    return "Network Started"

#Untuk nambah docker
@app.route('/tambahdocker')
def tambahdocker():
    tambahdockers.delay()
    return "docker berhasil ditambah"

@app.route('/formtambahlink')
def formtambahlink():
    return render_template('formtambahlink.html')

@app.route('/formlinkupdown')
def formlinkupdown():
    return render_template('formlinkupdown.html')

@app.route('/seelinkofswitch')
def seelinkofswitch():
    hasil = None
    if 'username' in session:
        username_form=str(session['username'])
        print username_form
        cur.execute("SELECT iduser FROM user WHERE
username = %s;", [username_form])
        hasil=cur.fetchall();
        for hasil1 in hasil:
```

```
            iduser=str(hasil1[0])
        print iduser
        cur.execute("SELECT linkofswitch.source,
userSource.username, linkofswitch.destination,
userDestination.username FROM  `linkofswitch` JOIN  `user`
userSource ON linkofswitch.source = userSource.iduser JOIN
`user` userDestination ON linkofswitch.destination =
userDestination.iduser WHERE source = %s OR destination =
%s", ([iduser], [iduser]))
        hasil=cur.fetchall();
        return render_template('seeLink.html', hasil =
hasil )
    error=None
    return render_template('login-gui.html', error=error)
'''
@app.route('/dockerStop')
def dockerStop():
'''

'''#########################################################
#########################################################
########################################'''

'''Method Get/Post'''

@app.route('/user', methods=['GET', 'POST'])
def user():
    '''if 'username' in session:
        username session = session['username']
        return render template('index-gui.html',
session user name=username session)'''
    return redirect(url_for('indexpage'))

@app.route('/login-gui', methods=['GET', 'POST'])
def login():
    print login
    error = None
    if 'username' in session:
        return redirect(url_for('indexpage'))
    if request.method == 'POST':
        username_form  = request.form['username']
        password_form  = request.form['password']
        cur.execute("SELECT COUNT(1) FROM user WHERE
username = %s;", [username_form]) # CHECKS IF USERNAME
EXSIST
        if cur.fetchone()[0]:
            cur.execute("SELECT pwd FROM user WHERE
username = %s;", [username_form]) # FETCH THE HASHED
PASSWORD
```

```
            for row in cur.fetchall():
                if password_form == row[0]:
                    session['username'] =
request.form['username']
                    return redirect(url_for('user'))
                else:
                    error = "Salah password!"
        else:
            error = "Anda belum terdaftar!"
    return render_template('login-gui.html', error=error)

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    print signup
    error = None
    if request.method == 'POST':
        username_form  = request.form['username']
        password_form  = request.form['password']
        cur.execute("SELECT COUNT(1) FROM user WHERE
username = %s;", [username_form]) # CHECKS IF USERNAME
EXSIST
        if cur.fetchone()[0]:
            error = "Username sudah digunakan!"
        else:
            cur.execute("INSERT INTO user(username, pwd)
VALUES(%s ,%s)", ([username_form],[password_form]))
            db.commit()
            error = "Berhasil Daftar!"
    return render_template('login-gui.html', error=error)

@app.route('/tambahlink', methods=['GET', 'POST'])
def tambahlink():
    source=session['username']
    dest=request.form['dest']
    status=1
    cur.execute("SELECT iduser FROM user WHERE username =
%s;", [source])
    hasil=cur.fetchall();
    for hasil1 in hasil:
            source_id=str(hasil1[0])
    cur.execute("SELECT iduser FROM user WHERE username =
%s;", [dest])
    hasil=cur.fetchall();
    for hasil1 in hasil:
            dest_id=str(hasil1[0])
    source_id_string="s"+str(source_id)
    dest_id_string="s"+str(dest_id)
    dest_id=int(dest_id)
    source_id=int(source_id)
```

```
    tambahlinks.delay(source_id_string,dest_id_string)
    cur.execute("INSERT INTO linkofswitch(source,
destination) VALUES(%s, %s);", ([source_id],[dest_id]))
    db.commit()
    return redirect(url_for('fetchTable'))

@app.route('/addDocker', methods=['GET', 'POST'])
def addDockers():
    error= None
    print index
    if request.method == 'POST':
        username_form = session['username']
        dockerName_form  = request.form['dockerName']
        dockerImage_form  = request.form['dockerImage']
        #dockerVar_form = request.form['dockerVar']

        #cari IDDocker
        cur.execute("SELECT iddocker FROM dockerhost ORDER
BY iddocker DESC LIMIT 1")
        hasil=cur.fetchall();
        for hasil1 in hasil:
            idDocker_form=str(int(str(hasil1[0]))+1)

        #cari UserID
        cur.execute("SELECT iduser FROM user WHERE
username = %s;", [username_form])
        hasil=cur.fetchall();
        for hasil1 in hasil:
            idUser_form=str(hasil1[0])
        #cur.execute("INSERT INTO
dockerhost(iduser,dockername,iddocker)
VALUES(%s,%s,%s)",([idUser_form],[dockerName_form],[idDock
er_form]) )
        ipAddress="10.0.0."+idDocker_form
        cur.execute("INSERT INTO
dockerhost(iduser,dockername,iddocker,jenisdocker,ip addre
ss)
VALUES(%s,%s,%s,%s,%s)",([idUser_form],[dockerName_form],[
idDocker_form], [dockerImage_form],[ipAddress]) )
        db.commit()

        #Nambah Docker di mininet
        tambahdockers.delay(idDocker_form,
"10.0.0."+idDocker_form+"/8", dockerImage_form,
"s"+idUser_form)

        return redirect(url_for('fetchTable'))

    #nyeluk shell/mininet script
```

```
    return "Under Construction"

@app.route('/upOrDown', methods=['GET', 'POST'])
def upOrDown():
    if request.method=='POST':
        dockerID_form = request.form['dockerID']
        mndocker="mn.d"+dockerID_form
        cur.execute("SELECT dockerstatus FROM dockerhost
WHERE iddocker = %s;", [dockerID_form])
        hasil=cur.fetchall();
        for hasil1 in hasil:
            dockerStatus_form=str(hasil1[0])

        if dockerStatus_form=='1':
            cur.execute("UPDATE dockerhost SET
dockerstatus=0 WHERE iddocker = %s;",[dockerID_form])
            db.commit()
            #popen stop docker
            output = Popen(['docker', 'stop', mndocker],
stdout=PIPE)
            output.communicate()
        else:
            cur.execute("UPDATE dockerhost SET
dockerstatus=1 WHERE iddocker = %s;",[dockerID_form])
            db.commit()
            #popen start docker
            output = Popen(['docker', 'start', mndocker],
stdout=PIPE)
            output.communicate()
        return redirect(url_for('fetchTable'))

@app.route('/deleteDocker', methods=['GET', 'POST'])
def deleteDocker():
    if request.method=='POST':
        dockerID_form = request.form['dockerID']
        mndocker="mn.d"+dockerID_form

        cur.execute("SELECT dockerstatus FROM dockerhost
WHERE iddocker = %s;", [dockerID_form])
        hasil=cur.fetchall();
        for hasil1 in hasil:
            dockerStatus_form=str(hasil1[0])

        if dockerStatus_form=='1':
            output = Popen(['docker', 'stop', mndocker],
stdout=PIPE)
            output.communicate()
        output = Popen(['docker', 'rm', mndocker],
stdout=PIPE)
```

```
        output.communicate()

        cur.execute("DELETE FROM dockerhost WHERE iddocker
= %s;",[dockerID_form])
        db.commit()
        return redirect(url_for('fetchTable'))


@app.route('/linkupdown', methods=['GET', 'POST'])
def linkupdown():
    if request.method=='POST':
        source=session['username']
        dest=request.form['dest']
        linkStatus=9
        cur.execute("SELECT iduser FROM user WHERE
username = %s;", [source])
        hasil=cur.fetchall();
        for hasil1 in hasil:
            source_id=str(hasil1[0])
        cur.execute("SELECT iduser FROM user WHERE
username = %s;", [dest])
        hasil=cur.fetchall();
        for hasil1 in hasil:
                dest_id=str(hasil1[0])
        source_id_string="s"+str(source_id)
        dest_id_string="s"+str(dest_id)
        #select status di link
        #cur.execute("SELECT status FROM linkofswitch
WHERE source = %s AND destination = %s;",
([source id],[dest id]))
        #hasil=cur.fetchall();
        #for hasil1 in hasil:
                #linkStatus=str(hasil1[0])
        #kalo up, pencet yg down
        #if linkStatus == '1':
        delLinks.delay(source_id_string,dest_id_string)
        cur.execute("DELETE FROM linkofswitch WHERE source
= %s AND destination = %s;", ([source_id],[dest_id]))
        db.commit()
        #vice versa

            #return "Link Not Found"

#linkUps.delay(source_id_string,dest_id_string)
            #cur.execute("UPDATE linkofswitch  SET
status=1 WHERE source = %s AND destination = %s;",
([source id],[dest id]))
            #db.commit()
        return "berhasil hapusnya"
```

```
@app.route('/seePort', methods=['GET', 'POST'])
def seePort():
    if request.method=='POST':
        dockerID_form = request.form['dockerID']
        mndocker="name=mn.d"+dockerID_form

        output = Popen(['docker', 'ps','-f',mndocker, '--
format', '"{{.Ports}}"'], stdout=PIPE)
        output = output.stdout.read()
        #return output
        return render_template('seePort.html', output =
output )
        #output.communicate()
        #return output.stdout.read()

'''##########################################################
##########################################################
########################################'''

'''Celery tasks'''

@celery.task
def myNetworks():
    lordy.myNetwork()

@celery.task
def tambahdockers(idDocker,ipDocker,imageDocker,switch):

lordy.tambahDocker(idDocker,ipDocker,imageDocker,switch)

@celery.task
def tesNets():
    lordy.lordyNet()

@celery.task
def addSwitchs(source_id):
    lordy.lordyAddSwitch(source_id)

@celery.task
def tambahlinks(source,dest):
    lordy.lordyAddLink(source,dest)

@celery.task
def delLinks(source,dest):
    lordy.lordyDelLink(source,dest)
```

```
#Other Method
def searchIDUser(params):
    cur.execute("SELECT iduser FROM user WHERE username =
%s;", [params])
    hasil=cur.fetchall();
    for hasil1 in hasil:
            result=str(hasil1[0])
    return result


app.secret_key = 'awankinton123'
if __name__ == '__main__':
    #tescelery()     #untuk ngestart jaringan
    app.run(debug=False,host= '0.0.0.0')
```

Kode Sumber 8.9 run-redis.sh

```
#!/bin/bash
if [ ! -d redis-stable/src ]; then
    curl -O http://download.redis.io/redis-stable.tar.gz
    tar xvzf redis-stable.tar.gz
    rm redis-stable.tar.gz
fi
cd redis-stable
make
src/redis-server
```

Kode Sumber 8.10 Perintah untuk menjalankan celery

```
sudo celery worker -A tes1.celery --loglevel=info
```

8.3 Virtual Data Center (Containernet)

Kode Sumber 8.11 lordy.py (Kode Containernet)

```
from mininet.net import Containernet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Docker, Node
from mininet.node import OVSKernelSwitch, UserSwitch
```

```
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

net = 0

def myNetwork():

    global net
    net = Containernet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='0.0.0.0',
                      protocol='tcp',
                      port=6633)

    info( '*** Add switches\n')
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    d4 = net.addDocker('d4', ip='10.0.0.4',
defaultRoute=None, dimage="ubuntu:trusty")
    d6 = net.addDocker('d6', ip='10.0.0.6',
defaultRoute=None, dimage="ubuntu:trusty")
    d1 = net.addDocker('d1', ip='10.0.0.1',
defaultRoute=None, dimage="ubuntu:trusty")
    d3 = net.addDocker('d3', ip='10.0.0.3',
defaultRoute=None, dimage="mysql/mysql-server:latest")
    d2 = net.addDocker('d2', ip='10.0.0.2',
defaultRoute=None, dimage="nazarpc/phpmyadmin:latest")
    d5 = net.addDocker('d5', ip='10.0.0.5',
defaultRoute=None, dimage="mysql/mysql-server:latest")

    info( '*** Add links\n')
    net.addLink(s1, s2)
    net.addLink(s2, s5)
    net.addLink(s5, s3)
    net.addLink(s3, s1)
    net.addLink(s3, s2)
```

```
    #net.addLink(s1, s4)
    net.addLink(s4, s3)
    net.addLink(s4, d1)
    net.addLink(s4, d2)
    net.addLink(s3, d3)
    net.addLink(s3, d4)
    net.addLink(s5, d5)
    net.addLink(s5, d6)

    net.removeLink(None,s4,s3)

    info("*** starting net")
    net.start()


    info("*** Kodingan Tambahan Lordy\n")
    #tambahDocker('11','10.0.0.11/8','ubuntu:trusty','s3')


    #CLI(net)
    #net.stop()

def tambahDocker(nomor,ipAddress,jenis,switchNo):
    global net
    nomor='d'+nomor
    d11 = net.addDocker(nomor, defaultRoute=None,
dimage=jenis)
    net.addLink(d11,net.get(switchNo),params1={"ip":
ipAddress}) #aslinya /8
    #netBuild() unneeded
    #/8 netmask, buat nandain IP jaringannya yg mana, ga
penting se

def netBuild():     #unneeded
    global net
    info( '*** Starting network versi LORDY\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s5').start([net.get('c0')])
    net.get('s4').start([net.get('c0')])
    net.get('s3').start([net.get('c0')])
    net.get('s1').start([net.get('c0')])
    net.get('s2').start([net.get('c0')])
    info( '*** Post configure switches and hosts\n')
```

```
def lordyNet():
    global net
    return str(net.values())
    #return 0

def lordyAddSwitch(switchName):
    global net
    s_user = net.addSwitch(switchName,
cls=OVSKernelSwitch)
    net.get(switchName).start([net.get('c0')])
    #net.addLink(s user,net.get('s1'))

def lordyAddLink(source,dest):
    global net
    net.addLink(net.get(source),net.get(dest))

def lordyDelLink(source,dest):
    global net
    net.removeLink(None,net.get(source),net.get(dest))



if __name__ == '__main__':
    setLogLevel( 'info' )
    #myNetwork()
```

8.4 Testing

Kode Sumber 8.12 stressTest.py (testing 5.2.1 dan 5.2.4)

```
from mininet.net import Containernet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Docker, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Containernet( topo=None,
                    build=False,
                    ipBase='10.0.0.0/8')
```

```
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='0.0.0.0',
                         protocol='tcp',
                         port=6633)
    d=[]
    info( '*** Add switches\n')

    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    for x in range(1000):
        xx = 'd'+str(x)
        ipd = '10.0.0.'+str(x)
        d.append(net.addDocker(xx, ip=ipd,
defaultRoute=None, dimage="ubuntu:trusty"))

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Kode Sumber 8.13 poxtop.py (testing 5.2.2 dan 5.2.5)

```
from mininet.net import Containernet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Docker, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.link import TCLink, Intf
```

```
from mininet.util import quietRun
from subprocess import call

net = 0

def myNetwork():

    global net

    net = Containernet( topo=None,
                    build=False,
                    ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                         controller=RemoteController,
                         ip='0.0.0.0',
                         protocol='tcp',
                         port=6633)

    info( '*** Add switches\n')

    net.addSwitch("s1", cls=OVSKernelSwitch)
    net.get("s1").start([net.get('c0')])
    print "..done adding switches"
    print "..adding host"
    d1 = net.addDocker('d1', ip='10.0.0.1',
defaultRoute=None, dimage="ubuntu:trusty")
    d2 = net.addDocker('d1', ip='10.0.0.2',
defaultRoute=None, dimage="ubuntu:trusty")


    net.addLink(d1,net.get("s1"))
    net.addLink(d2,net.get("s1"))

    info("*** starting net")
    net.start()

    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Kode Sumber 8.14 poxtopology.py (testing 5.2.3 dan 5.2.6)

```
from mininet.net import Containernet
from mininet.node import Controller, RemoteController,
OVSController
from mininet.node import CPULimitedHost, Docker, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Containernet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='0.0.0.0',
                      protocol='tcp',
                      port=6633)

    d=[]
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    for x in range(100):
        xx = 'd'+str(x)
        ipd = '10.0.0.'+str(x)
        d.append(net.addDocker(xx, ip=ipd,
defaultRoute=None, dimage="ubuntu:trusty"))

    info( '*** Add links\n')
    for y in range(100):
        if y<=99:
            net.addLink(s1, d[y])
            print y

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()
```

```
    info( '*** Starting switches\n')
    net.get('s1').start([c0])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Kode Sumber 8.15 Perintah MySQL (testing 5.3.7)

```
CREATE DATABASE testing;
Use testing;
CREATE TABLE users (id INT);
Create procedure addUsers()
 begin
   declare i int default 1;
   while (i <= 1000) do
     insert into users (id) values (i);
     set i=i+1;
   end while;
call addUsers();
```

Kode Sumber 8.16 Perintah *sysbench* (testing 5.3.8)

```
sysbench --test=oltp --oltp-table-size=10000 \
--db-driver=mysql --mysql-db=testing --mysql-user=root \
--mysql-password=asu --mysql-host=127.0.0.1 prepare

sysbench \
    --test=oltp \
    --oltp-read-only=off \
    --oltp-table-size=10000 \
    --oltp-test-mode=complex \
    --num-threads=40 \
    --max-requests=0 \
    --max-time=600 \
    --mysql-db=testing \
    --mysql-user=root \
    --mysql-password=1234 \
    --mysql-host=127.0.0.1 \
```

```
    run
```

Kode Sumber 8.17 Perintah *sysbench* (testing 5.3.9)

```
sysbench --test=fileio --file-total-size=8G prepare

sysbench --test=fileio --file-total-size=8G --file-
test-mode=rndrw --init-rng=on --max-time=300 --max-
requests=0 run

sysbench --test=fileio --file-total-size=150G
cleanup
```

Kode Sumber 8.18 Perintah *MySqlSlap* (testing 5.3.10)

```
sudo mysqlslap --user=root --password=123456 --
host=localhost  --concurrency=50 --number-of-
queries=1000 --iterations=10 --auto-generate-sql --
verbose
```

BIODATA PENULIS

Muhammad Fikri Alauddin merupakan anak pertama dari pasangan Bapak Budi Hariono dan Ibu Imas Parnika Winata. Lahir di Jember pada tanggal 30 Juli 1995. Penulis menempuh pendidikan formal dimulai dari TK Adh-Dhuha Jember (199-2001), SDN Jember Lor 3 (2001-2007), SMPN 3 Jember (2007-2013), SMAN 1 Jember dan sesudah lulus dari SMAN 1 Jember melanjutkan menimba ilmu di jurasan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember, Surabaya (2013-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2014-2016) sebagai Staf Departemen Hubungan Luar (2014-2015) dan Staf Ahli Departemen Hubungan Luar (2016-2016) Teknik Computer-Informatika. Selain itu, juga memiliki pengalaman kepanitiaan, diantaranya sebagai Staf Dana dan Usaha Schematics 2014 dan 2015. Penulis juga menyukai kegiatan sosial. Penulis memiliki hobi bermain game terutama game online dan menyukai hal baru. Penulis dapat dihubungi melalui email: lostman.fikri@gmail.com.